Оглавление

[Нормализация реляционной БД. Основные нормальные формы: 1НФ, 2НФ, 3НФ, НФБК. Дополнительно изучить: 4НФ, 5НФ. 6НФ. Якорная модель. 2](#_Toc198806177)

[Типы связей, один к одному, многие ко многим т.д. 10](#_Toc198806178)

[Денормализация реляционной БД. 12](#_Toc198806179)

[Понятие "Хранилище данных" 14](#_Toc198806180)

[OLAP, OLTP, в чём отличие и почему OLTP не подходит для аналитических систем 17](#_Toc198806181)

[Что такое ETL и для чего он нужен. Основные проблемы. Отличие ETL от ELT подхода. 21](#_Toc198806182)

[Dimension model. Какие данные должны хранится в Dimension. 23](#_Toc198806183)

[Fact, какие данные должны хранится в Fact. 27](#_Toc198806184)

[Меры (measures), типы мер. 29](#_Toc198806185)

[Гранулярность данных. 32](#_Toc198806186)

[Что такое суррогатные ключи, для чего нужны и какие приемущества они приносят в DWH. 33](#_Toc198806187)

[Принцип построения звезды и снежинки, различия и недостатки. 35](#_Toc198806188)

[SCD, зачем нужно и какие виды бывают. 37](#_Toc198806189)

[Инкрементальный подход загрузки данных в DWH. 38](#_Toc198806190)

[Витрина данных. Зависимые и независимые витрины. 41](#_Toc198806191)

[Какие области хранилища существуют и для чего каждая из них нужна 44](#_Toc198806192)

## Нормализация реляционной БД. Основные нормальные формы: 1НФ, 2НФ, 3НФ, НФБК. Дополнительно изучить: 4НФ, 5НФ. 6НФ. Якорная модель.

Что такое нормализация?

Нормализация — это метод проектирования базы данных, направленный на улучшение её структуры путём устранения повторений и повышения целостности данных. Основная цель процесса нормализации состоит в предотвращении возникновения различных типов аномалий, таких как проблемы при обновлении, удалении или вставке данных.

Как работают нормальные формы?

Процесс нормализации делится на **этапы**, называемые **нормальными формами**. Они последовательно решают конкретные проблемы организации данных в таблицах. Рассмотрим подробнее наиболее важные формы.

Зачем соблюдать нормы нормализации?

Соблюдение правил нормализации помогает избегать следующих проблем:

* Избыточность данных (ненужное дублирование информации);
* Потеря согласованности данных (при изменении данных в одном месте приходится менять их везде);
* Трудности при поддержке целостности данных (сложнее поддерживать взаимосвязанные таблицы);
* Увеличение размера базы данных (лишние строки и лишние столбцы занимают много места).

Следуя принципам нормализации, разработчики создают надёжные и эффективные схемы хранения данных, облегчающие работу с базой данных и уменьшающие риск ошибок.

Это упрощает администрирование и снижает нагрузку на систему при выполнении операций чтения-записи данных.

1. Первая нормальная форма (1NF)

Первая нормальная форма означает, что база данных организована таким образом, чтобы каждое поле было атомарным (неделимым). Если поле содержит список значений или сложный объект, оно нарушает требования 1NF.

Атомарные значения, в контексте баз данных и программирования, означают неделимые, минимальные единицы данных. Это могут быть отдельный атрибут в базе данных, который нельзя разделить на меньшие части без потери смысла.

**Требования 1NF:**

* Каждый атрибут (столбец) таблицы содержит одиночные значения.
* Значения в каждом столбце не делятся на составляющие элементы.
* Записи идентифицируются уникальным значением (первичный ключ).

**Пример нарушения 1NF:**Допустим, у нас есть таблица сотрудников с такими данными:

| ID\_сотрудника | ФИО | Контакты |

|---------------|------------------|------------------------|

| 1 | Иван Петров | +7(987)-654-3210, ivan@mail.ru |

| 2 | Мария Иванова | maria@gmail.com |

Значение в столбце «Контакты» нарушено требованиями 1NF, потому что там содержится сразу два элемента (телефон и email), которые следовало бы разделить на разные столбцы.

**Исправление:** Разделение сложного атрибута на атомарные значения:

| ID\_сотрудника | ФИО | Телефон | Email |

|---------------|----------------|-----------------|-----------------|

| 1 | Иван Петров | +7(987)-654-3210 | ivan@mail.ru |

| 2 | Мария Иванова | NULL | maria@gmail.com |

Теперь каждое поле представляет собой единичное значение, удовлетворяя условиям 1NF.

2. Вторая нормальная форма (2NF)

Переход ко второй нормальной форме обеспечивает устранение **частичных зависимостей**: ситуации, когда некоторые неключевые поля зависят лишь частично от первичного ключа.

**Требования 2NF:**

* Таблица должна быть в 1NF.
* Все неключевые атрибуты должны полностью зависеть от всего первичного ключа.

Частичная зависимость возникает тогда, когда в качестве первичного используется составной ключ (например, комбинация двух полей), а какое-то неключевое поле зависит лишь от одного компонента этого ключа.

**Пример нарушения 2NF:**Рассмотрим следующую таблицу заказов:

| КодЗаказа | ДатаЗаказa | КлиентID | СуммаЗаказa |

|-----------|------------|----------|-------------|

| Z001 | 2023-01-01 | C001 | 1000 |

| Z002 | 2023-01-02 | C002 | 1500 |

Предположим, что мы используем составной ключ ("КодЗаказа", "КлиентID"). Однако сумма заказа зависит исключительно от "КодЗаказа", а не от обоих элементов вместе. Следовательно, имеет место частичная зависимость суммы от части ключа.

**Исправление:** Нужно разбить такую таблицу на две, чтобы устранить частичную зависимость:

Таблица Заказы:

| КодЗаказа | ДатаЗаказа | СуммаЗаказа |

|-----------|------------|-------------|

| Z001 | 2023-01-01 | 1000 |

| Z002 | 2023-01-02 | 1500 |

Таблица Клиенты:

| КлиентID | ИмяКлиента |

|----------|------------|

| C001 | Петр Петров|

| C002 | Анна Иванова|

Теперь вся информация хранится отдельно, и нарушений требований 2NF больше нет.

3. Третья нормальная форма (3NF)

Третья нормальная форма устраняет проблему **транзитивных зависимостей** — ситуаций, когда один неключевой атрибут зависит от другого неключевого атрибута, а не напрямую от первичного ключа.

**Требования 3NF:**

* Таблица должна быть в 2NF.
* Неключевые атрибуты не должны зависеть друг от друга.

Транзитивная зависимость возникает, когда некий неключевой атрибут влияет на другой неключевой атрибут, минуя первичный ключ.

Транзитивная связь (транзитивная зависимость) — это особый вид зависимости в отношениях между атрибутами внутри таблицы базы данных, при котором изменение одного атрибута косвенным образом влияет на другой атрибут, минуя прямой контакт с ключевым полем. Говоря проще, эта связь показывает, что изменение в одном поле вызывает цепочку изменений в других полях, причём без прямого влияния на ключевой атрибут.

**Пример нарушения 3NF:**Представьте себе таблицу товаров с категориями:

| Артикул | Название товара | Категория | Описание категории |

|---------|-----------------|-----------|--------------------|

| T001 | Телевизор LG | Электроника| Высококачественная техника |

| T002 | Кофемашина Bosch| Кухонные приборы| Удобство на кухне |

Очевидно, что «Описание категории» зависит не от артикула товара, а от самой категории. Здесь присутствует транзитивная зависимость, нарушающая правила 3NF.

**Исправление:** Необходимо выделить отдельную таблицу для описания категорий:

Таблица Товары:

| Артикул | Название товара | Категория |

|---------|-----------------|-----------|

| T001 | Телевизор LG | Электроника|

| T002 | Кофемашина Bosch| Кухонные приборы|

Таблица Категории:

| Категория | Описание категории |

|-----------|--------------------|

| Электроника| Высококачественная техника |

| Кухонные приборы| Удобство на кухне |

Теперь таблицы нормализованы, и проблема транзитивной зависимости устранена.

4. Нормальная форма Бойса–Кодда (BCNF)

#### **Нормальная форма Бойса-Кодда (НФБК) (частная форма третьей нормальной формы)**

Определение 3НФ не совсем подходит для следующих отношений:  
1) отношение имеет два или более потенциальных ключа;  
2) два и более потенциальных ключа являются составными;  
3) они пересекаются, т.е. имеют хотя бы один общий атрибут.  
  
Для отношений, имеющих один потенциальный ключ (первичный), НФБК является 3НФ.  
  
Отношение находится в НФБК, когда каждая нетривиальная и неприводимая слева функциональная зависимость обладает потенциальным ключом в качестве детерминанта.  
  
Предположим, рассматривается отношение, представляющее данные о бронировании стоянки на день:

|  |  |  |  |
| --- | --- | --- | --- |
| Номер стоянки | Время начала | Время окончания | Тариф |
| 1 | 09:30 | 10:30 | Бережливый |
| 1 | 11:00 | 12:00 | Бережливый |
| 1 | 14:00 | 15:30 | Стандарт |
| 2 | 10:00 | 12:00 | Премиум-В |
| 2 | 12:00 | 14:00 | Премиум-В |
| 2 | 15:00 | 18:00 | Премиум-А |

Тариф имеет уникальное название и зависит от выбранной стоянки и наличии льгот, в частности:

* «Бережливый»: стоянка 1 для льготников
* «Стандарт»: стоянка 1 для не льготников
* «Премиум-А»: стоянка 2 для льготников
* «Премиум-B»: стоянка 2 для не льготников.

Таким образом, возможны следующие составные первичные ключи: {Номер стоянки, Время начала}, {Номер стоянки, Время окончания}, {Тариф, Время начала}, {Тариф, Время окончания}.  
  
Отношение находится в 3НФ. Требования второй нормальной формы выполняются, так как все атрибуты входят в какой-то из потенциальных ключей, а неключевых атрибутов в отношении нет. Также нет и транзитивных зависимостей, что соответствует требованиям третьей нормальной формы. Тем не менее, существует функциональная зависимость Тариф → Номер стоянки, в которой левая часть (детерминант) не является потенциальным ключом отношения, то есть отношение не находится в нормальной форме Бойса — Кодда.  
  
Недостатком данной структуры является то, что, например, по ошибке можно приписать тариф «Бережливый» к бронированию второй стоянки, хотя он может относиться только к первой стоянки.  
  
Можно улучшить структуру с помощью декомпозиции отношения на два и добавления атрибута **Имеет льготы**, получив отношения, удовлетворяющие НФБК (подчёркнуты атрибуты, входящие в первичный ключ.):  
  
**Тарифы**

|  |  |  |
| --- | --- | --- |
| Тариф | Номер стоянки | Имеет льготы |
| Бережливый | 1 | Да |
| Стандарт | 1 | Нет |
| Премиум-А | 2 | Да |
| Премиум-В | 2 | Нет |

**Бронирование**

|  |  |  |
| --- | --- | --- |
| Тариф | Время начала | Время окончания |
| Бережливый | 09:30 | 10:30 |
| Бережливый | 11:00 | 12:00 |
| Стандарт | 14:00 | 15:30 |
| Премиум-В | 10:00 | 12:00 |
| Премиум-В | 12:00 | 14:00 |
| Премиум-А | 15:00 | 18:00 |

# **Сравнение 3НФ (Третьей нормальной формы) и НФБК (Нормальной формы Бойса-Кодда)**

## **Основные различия**

| **Критерий** | **3НФ (Третья нормальная форма)** | **НФБК (Нормальная форма Бойса-Кодда)** |
| --- | --- | --- |
| **Определение** | Таблица находится в 3НФ, если она во 2НФ и нет транзитивных зависимостей неключевых атрибутов от первичного ключа | Таблица находится в НФБК, если детерминанты всех функциональных зависимостей являются потенциальными ключами |
| **Требования** | Устранение транзитивных зависимостей неключевых атрибутов от первичного ключа | Устранение всех зависимостей атрибутов от неключевых атрибутов |
| **Строгость** | Менее строгая форма нормализации | Более строгая форма нормализации |
| **Охват** | Частный случай НФБК | Обобщение 3НФ, охватывает больше случаев |

## **Подробное объяснение**

### **3НФ (Третья нормальная форма)**

Таблица находится в 3НФ, если:

1. Она находится во 2НФ (нет частичных зависимостей неключевых атрибутов от составного первичного ключа)
2. Нет транзитивных зависимостей неключевых атрибутов от первичного ключа

**Пример нарушения 3НФ**:

Copy

Download

Заказ (НомерЗаказа, Дата, КодКлиента, ИмяКлиента, АдресКлиента)

Здесь ИмяКлиента и АдресКлиента зависят от КодКлиента, а не напрямую от НомерЗаказа (первичного ключа).

### **НФБК (Нормальная форма Бойса-Кодда)**

Таблица находится в НФБК, если:

1. Она находится в 3НФ
2. Каждый детерминант (атрибут, от которого функционально зависит другой атрибут) является потенциальным ключом

**Пример нарушения НФБК (но соответствие 3НФ)**:

Copy

Download

Студент\_Курс (НомерСтудента, КодКурса, Преподаватель, Кафедра)

Предположим:

* Один преподаватель ведет один курс
* Один преподаватель привязан к одной кафедре

Здесь:

* КодКурса → Преподаватель (детерминант - не потенциальный ключ)
* Преподаватель → Кафедра (транзитивная зависимость)

Таблица в 3НФ, но не в НФБК, так как КодКурса не является потенциальным ключом.

## **Когда таблица в 3НФ, но не в НФБК?**

Это происходит, когда:

1. В таблице есть составной первичный ключ
2. Один из атрибутов составного ключа функционально определяет неключевой атрибут

## Четвёртая нормальная форма (4NF)

Четвертая нормальная форма ориентирована на устранение многозначных зависимостей. Многозначная зависимость появляется, когда одно отношение (связь) порождает несколько независимых списков значений.

**Что это значит?**

Многозначная зависимость возникает, когда набор атрибутов образует независимые множества значений, зависящих от одного и того же набора атрибутов. Простым языком: одна запись может иметь несколько множественных значений разных характеристик, которые нельзя однозначно связать друг с другом.

**Пример нарушения 4NF:**Возьмем пример студентов и курсов:

| СтудентID | Курс | Преподаватель |

|-----------|------|---------------|

| S001 | Математика | Иванов |

| S001 | История | Сидоров |

| S002 | Физика | Кузнецов |

Обратите внимание, что студент может посещать разные курсы и каждый курс ведёт отдельный преподаватель. Между этими тремя характеристиками (Студент, Курс, Преподаватель) существует многозначная зависимость, нарушая 4NF.

**Исправление:**Для соблюдения 4NF необходимо исключить такие связи, разнеся их по отдельным таблицам:

Таблица Студентов:

| СтудентID | Имя студента |

|-----------|-------------|

| S001 | Алексей |

| S002 | Екатерина |

Таблица Курсов:

| Курс | Преподаватель |

|------|---------------|

| Математика | Иванов |

| История | Сидоров |

| Физика | Кузнецов |

Таблица Посещаемости:

| СтудентID | Курс |

|-----------|------|

| S001 | Математика |

| S001 | История |

| S002 | Физика |

## Пятая нормальная форма (5NF)

Пятая нормальная форма, также известная как проектируемая декомпозиция, предназначена для избавления от сложных зависимостей, называемых соединительными зависимостями. Соединительная зависимость проявляется, когда существует зависимость между несколькими наборами атрибутов, образующих множество сочетаний.

**Цель 5NF:**Избавиться от ситуаций, когда сложная структура отношений затрудняет хранение и обработку данных. Важно отметить, что применение пятой нормальной формы часто бывает редким и применяется в специализированных случаях.

**Пример нарушения 5NF:**Представим три типа сотрудников: продавцы, консультанты и инженеры. У продавцов есть определённые товары, консультантов привлекают к разным проектам, а инженеров используют в рамках конкретных технологий. Возможна такая схема:

| Продавец | Консультант | Инженер | Проект |

|----------|-------------|---------|--------|

| A | B | C | P1 |

| D | E | F | P2 |

Эти роли пересекаются случайным образом, создавая сложную комбинаторную схему, которую сложно разложить на элементарные отношения.

**Исправление:**Чтобы привести такую схему к 5NF, необходимо выявить базовые взаимоотношения и разложить таблицу на несколько простых структур:

Таблица проектов-продавцов:

| Продавец | Проект |

|----------|--------|

| A | P1 |

| D | P2 |

Таблица проектов-консультантов:

| Консультант | Проект |

|-------------|--------|

| B | P1 |

| E | P2 |

Таблица проектов-инженеров:

| Инженер | Проект |

|---------|--------|

| C | P1 |

| F | P2 |

Каждая новая таблица теперь описывает простую однозначную связь, устраняя необходимость сложной конструкции исходной таблицы.

## Шестая нормальная форма (6NF)

Шестая нормальная форма — это дальнейшее развитие концепции нормализации, которое применяется крайне редко и направлено на полное исключение любой возможной сложности и зависимости между атрибутами. По сути, это предельный уровень нормализации, когда таблицы становятся настолько мелкими, что практически теряют практическое значение.

**Практическая значимость:**Обычно шестая нормальная форма рассматривается скорее теоретически, чем применительно к реальной практике разработки баз данных. Достижение такой степени детализации чаще всего излишне усложняет архитектуру системы и делает её менее эффективной.

## Якорная модель (Anchor Model)

**Якорная модель** —тем, что она сосредоточена на неизменяемых сущностях (так называемых "якорях"), вокруг которых группируются различные характеристики и события.

### Особенности якорной модели:

* **Якоря** — фундаментальные сущности, представляющие реальные объекты мира (например, человек, продукт, компания). Их главное свойство — они остаются неизменными во времени.
* **Атрибуты** — динамические свойства якорей, которые могут меняться с течением времени (например, имя человека, адрес фирмы).
* **Связи** — отображают связи между якорями (например, кто кому подчиняется, какая фирма производит какой товар).
* **Роли** — определяют смысл атрибутов и связей в конкретном контексте.

Основная идея якорной модели — возможность легко адаптироваться к изменениям предметной области и структуре данных, сохраняя при этом целостную картину исторических изменений.

Якорная модель (Anchor Modeling) — это особый подход к проектированию баз данных, который идеально подходит для часто меняющихся структур данных. Давайте разберём его на простых примерах.

## **Ключевые компоненты**

1. **Якоря (Anchors)**
   * Неизменяемые сущности (например, "Клиент", "Заказ")
   * Содержат только ID и дату создания
   * Пример: Клиент(ID, Дата\_создания)
2. **Атрибуты (Attributes)**
   * Характеристики якорей, которые могут меняться
   * Каждый атрибут хранит историю изменений
   * Пример: Имя\_клиента(Клиент\_ID, Имя, Дата\_начала, Дата\_окончания)
3. **Связи (Ties)**
   * Отношения между якорями
   * Могут быть историческими (меняться со временем)
   * Пример: Клиент\_Заказ(Клиент\_ID, Заказ\_ID, Дата\_связи)
4. **Узлы (Knots)**
   * Общие справочники для часто используемых значений
   * Пример: Тип\_оплаты(Код, Значение)

### Преимущества якорной модели:

* Возможность легко отслеживать изменения во времени (различные версии объектов сохраняются автоматически).
* Упрощённая поддержка эволюции данных (легко добавлять новые типы данных, удалять старые).
* Независимость моделей от конкретной реализации СУБД.

Однако, несмотря на преимущества, якорная модель считается довольно сложной в понимании и применении, особенно для небольших и средних приложений. Её использование оправдано преимущественно в крупных проектах, где важна гибкость архитектуры и отслеживание истории изменений данных.

Подведём итог:

* **4NF** решает проблему многозначных зависимостей.
* **5NF** избавляет от соединений сложных взаимоотношений.
* **6NF** доводит концепцию нормализации до теоретического идеала.
* **Якорная модель** предлагает совершенно иной взгляд на проектирование схем данных, фокусируясь на стабильности и историчности данных.

Транзитивная связь (транзитивная зависимость) — это особый вид зависимости в отношениях между атрибутами внутри таблицы базы данных, при котором изменение одного атрибута косвенным образом влияет на другой атрибут, минуя прямой контакт с ключевым полем. Говоря проще, эта связь показывает, что изменение в одном поле вызывает цепочку изменений в других полях, причём без прямого влияния на ключевой атрибут.

### Формальное определение:

Транзитивная зависимость в реляционной теории данных возникает, когда один неключевой атрибут зависит от другого неключевого атрибута, а не напрямую от первичного ключа.

Формально выражаясь, если есть зависимость A→B*A*→*B* и B→C*B*→*C*, но при этом нет непосредственной зависимости A→C*A*→*C*, это создаёт транзитивную зависимость.

### Почему транзитивные связи нежелательны?

Проблема транзитивных связей заключается в том, что они приводят к увеличению риска потери целостности данных и повышению вероятности появления противоречивых или устаревших сведений. Транзитивные зависимости делают базу данных уязвимой к ошибкам синхронизации и снижают эффективность операций модификации данных.

### Пример транзитивной связи:

Допустим, у нас есть таблица сотрудников, включающая информацию о сотрудниках и подразделениях, в которых они работают:

| EmployeeID | DepartmentID | ManagerID | SalaryLevel |

|------------|--------------|-----------|-------------|

| EMP001 | DEPT001 | MAN001 | High |

| EMP002 | DEPT002 | MAN002 | Medium |

Причём подразделения организованы следующим образом:

| DepartmentID | Location |

|--------------|--------------|

| DEPT001 | Москва |

| DEPT002 | Санкт-Петербург|

И менеджеры подразделений тоже распределены по городам:

| ManagerID | City |

|-----------|--------------|

| MAN001 | Москва |

| MAN002 | Санкт-Петербург|

В данном примере получается следующая цепь зависимостей:

* EmployeeID → DepartmentID
* DepartmentID → Location
* ManagerID → City

Но обратите внимание, что город расположения менеджера фактически зависит от подразделения, в котором работает сотрудник. Поэтому образуется транзитивная зависимость:

EmployeeID→DepartmentID,DepartmentID→CityEmployeeID→DepartmentID,DepartmentID→City

Таким образом, изменяется город, связанный с сотрудником, не напрямую, а через подразделение, в котором он работает.

### Проблема транзитивной зависимости:

Такой подход плох тем, что внесение изменений становится затруднительным. Например, если изменить город подразделения, изменится и город менеджера, даже если физически менеджер никуда не переезжал. Или, наоборот, перемещение менеджера потребует внесения массы исправлений вручную.

### Устранение транзитивных зависимостей:

Правильным решением станет нормализация базы данных до третьей нормальной формы (3NF), где все транзитивные зависимости исключаются. Для нашего примера правильным решением будет создание отдельной таблицы, хранящей связь между департаментами и менеджерами, чтобы избежать указанной цепи зависимостей.

Итак, сущность транзитивной связи проста: она свидетельствует о нарушении принципа независимости атрибутов, приводящем к проблемам целостности и эффективности данных.

### **Строковые и колоночные базы данных**

#### **Строковые базы данных (Row-oriented databases)**

**Строковые базы данных** — это традиционные базы данных, в которых данные хранятся в виде строк (записей). Каждая строка содержит все атрибуты (столбцы) для одной сущности. Строковые базы данных оптимизированы для операций чтения и записи отдельных строк.

**Преимущества строковых баз данных:**

1. **Быстрое чтение и запись отдельных строк**: Строковые базы данных оптимизированы для операций с отдельными строками, что делает их подходящими для транзакционных систем (OLTP).
2. **Простота использования**: Строковые базы данных широко распространены и поддерживаются большинством СУБД.
3. **Поддержка сложных запросов**: Строковые базы данных хорошо подходят для выполнения сложных запросов, которые требуют доступа к нескольким атрибутам одной сущности.

**Примеры строковых баз данных:**

* **Oracle Database**
* **Microsoft SQL Server**
* **MySQL**
* **PostgreSQL**

#### **Колоночные базы данных (Column-oriented databases)**

**Колоночные базы данных** — это базы данных, в которых данные хранятся в виде столбцов (колонок). Каждая колонка содержит значения одного атрибута для всех сущностей. Колоночные базы данных оптимизированы для операций чтения и анализа больших объемов данных.

**Преимущества колоночных баз данных:**

1. **Быстрое чтение больших объемов данных**: Колоночные базы данных оптимизированы для операций чтения больших объемов данных, что делает их подходящими для аналитических систем (OLAP).
2. **Эффективное сжатие данных**: Колоночные базы данных могут эффективно сжимать данные, так как значения в одной колонке часто повторяются.
3. **Поддержка параллельной обработки**: Колоночные базы данных хорошо подходят для параллельной обработки данных, что позволяет выполнять сложные аналитические запросы.

**Примеры колоночных баз данных:**

* **Apache Cassandra**
* **Apache HBase**
* **Amazon Redshift**
* **Google BigQuery**

### **Заключение**

Строковые и колоночные базы данных имеют свои преимущества и недостатки. Строковые базы данных оптимизированы для операций с отдельными строками и подходят для транзакционных систем, в то время как колоночные базы данных оптимизированы для операций чтения больших объемов данных и подходят для аналитических систем. Выбор типа базы данных зависит от конкретных требований и задач.

## Типы связей, один к одному, многие ко многим т.д.

Типы связей играют важнейшую роль в проектировании баз данных, определяя характер взаимодействия между различными объектами. Когда речь идет о взаимодействии таблиц в реляционных системах управления базами данных (СУБД), выделяют три основных типа связей: «один к одному», «один ко многим» и «многие ко многим». Позвольте объяснить их детально, словно я готовлюсь рассказать вам своим голосом перед экзаменационной комиссией.

## Один к одному (One-to-one)

**Один к одному** обозначает связь, при которой одна запись в одной таблице соответствует ровно одной записи в другой таблице, и наоборот. Такая связь встречается реже остальных и зачастую обусловлена ситуацией, когда две тесно связанные сущности имеют схожий жизненный цикл, но требуют разделения по организационным причинам.

**Особенности One-to-one:**

* Часто реализуется путем введения внешнего ключа (foreign key) в одну из таблиц, ссылающегося на уникальный идентификатор (primary key) другой таблицы.
* Может использоваться для расширения возможностей таблицы или для разделения большой таблицы на меньшие компоненты.

**Примеры:**

* Пользовательская учетная запись и профиль пользователя. Хотя аккаунт пользователя может существовать сам по себе, иногда полезно хранить личные данные отдельно от общей информации о регистрации.
* Автомобиль и регистрационные номера. У автомобиля может быть только один зарегистрированный номер, и каждый регистрационный номер привязан строго к одному автомобилю.

## Один ко многим (One-to-many)

Наиболее распространённый тип связи — **«один ко многим»**. Этот вид связи подразумевает ситуацию, когда одна запись в главной таблице связана с множеством записей в подчинённой таблице, однако каждая запись подчиненной таблицы относится только к одной записи основной таблицы.

**Особенности One-to-many:**

* Реализуется добавлением внешнего ключа в подчинённую таблицу, который ссылается на первичный ключ главной таблицы.
* Одна сторона отношений называется родительской, другая — дочерней.

**Примеры:**

* Клиент и заказы. У клиента может быть несколько заказов, но каждый заказ принадлежит единственному клиенту.
* Автор и книги. Автор может написать множество книг, но каждая книга написана одним автором.

## Многие ко многим (Many-to-many)

Самый сложный тип связи — **«многие ко многим»**. Он отражает ситуацию, когда записи одной таблицы могут относиться к нескольким записям другой таблицы, и наоборот. Такие отношения встречаются повсеместно и отражают комплексные связи между объектами реального мира.

**Особенности Many-to-many:**

* Обычно реализуются посредством дополнительной промежуточной таблицы (таблицы соединения), содержащей внешние ключи обеих сторон.
* Позволяет двум объектам взаимодействовать свободно, без ограничений на количество взаимных ссылок.

**Примеры:**

* Учащиеся и предметы. Учащийся может изучать несколько предметов, а предмет изучается разными учащимися.
* Актёры и фильмы. Актёр может сниматься в нескольких фильмах, а фильм может включать многих актёров.

## Дополнительные аспекты:

Помимо указанных трех основных типов, существуют другие виды связей, хотя они используются значительно реже:

* **Self-referencing relationship** (самоссылающаяся связь) — ситуация, когда запись в таблице ссылается сама на себя. Например, иерархия сотрудников в компании, где старший менеджер может управлять несколькими младшими сотрудниками.
* **Weak entity set** (слабые сущности) — объекты, существование которых обусловлено существованием других объектов. Примером служит детская медицинская карта, которая зависит от существования ребёнка.

## Денормализация реляционной БД.

## **1. Что такое денормализация?**

**Определение:**  
Денормализация — это намеренное отклонение от нормальных форм (1НФ, 2НФ, 3НФ, НФБК) с целью **ускорения выполнения запросов** за счёт **допущения избыточности данных**.

### **Когда применяется?**

✅ В **хранилищах данных (DWH)** и аналитических системах (OLAP)  
✅ Для **ускорения сложных JOIN-запросов**  
✅ В отчетных системах, где важна **скорость выборки**, а не целостность

## **2. Зачем денормализовать DWH?**

| **Проблема нормализованной БД** | **Решение денормализацией** |
| --- | --- |
| Много JOIN-ов → медленные запросы | Данные хранятся в одной таблице → меньше JOIN-ов |
| Сложные агрегации (SUM, AVG) | Предварительно вычисленные поля (например, total\_sales) |
| Частые запросы к справочникам | Встраивание справочных данных в фактовую таблицу |

## **Плюсы и минусы денормализации**

### **✔ Преимущества**

* **Ускорение запросов** (меньше JOIN-ов)
* **Упрощение SQL** (меньше подзапросов)
* **Лучшая производительность для отчетов**

### **❌ Недостатки**

* **Избыточность данных** (больше места)
* **Сложность обновлений** (надо менять данные в нескольких местах)
* **Риск несогласованности** (если данные обновляются не везде)

Денормализация базы данных — это целенаправленное отклонение от принципов нормализации ради улучшения производительности и удобства использования системы. Процесс денормализации предполагает намеренное введение избыточности данных для достижения большей скорости выборки и снижения нагрузки на сервер базы данных.

## Причины денормализации

Иногда строгая нормализация, позволяющая уменьшить избыточность и повысить целостность данных, оказывается неэффективной в плане быстродействия. Вот несколько примеров, когда денормализация полезна:

* **Повышение производительности запросов.** Нормализованные базы данных требуют многократных объединений (JOIN'ов) таблиц для извлечения данных, что замедляет выполнение запросов.
* **Оптимизация аналитики.** Некоторые приложения требуют быстрого предоставления отчетов или выполнения сложных аналитических запросов, и отказ от объединения таблиц ускоряет получение результатов.
* **Обработка большого объема данных.** В высоконагруженных системах с большими объемами данных объединение таблиц сильно увеличивает затраты ресурсов сервера.

## Способы денормализации

Существует несколько методов, которые позволяют снизить влияние нормализации на производительность системы:

### 1. Повторяющиеся данные (избыточность):

Этот способ предусматривает повторение некоторых данных в нескольких местах базы данных. Несмотря на увеличение объёма хранимых данных, это улучшает скорость выборки.

**Пример:** Допустим, у нас есть таблица сотрудников и таблица отделов. В таблице сотрудников изначально содержатся только уникальные коды отделов, и для получения названия отдела требуется объединение с таблицей отделов. Чтобы ускорить выборку, название отдела можно добавить прямо в таблицу сотрудников, создав небольшую избыточность.

### 2. Хранение производных данных:

Некоторые вычисления могут быть выполнены заранее и сохранены в виде готовых значений. Это существенно уменьшает вычислительную нагрузку на сервер.

**Пример:** Предположим, в магазине онлайн-продаж нам нужно быстро получать общую сумму покупок каждого покупателя. Вместо постоянного подсчета продаж можно периодически сохранять предварительно рассчитанный суммарный показатель.

### 3. Создание агрегированных таблиц:

Агрегированные таблицы содержат заранее подготовленные итоги, сводки и отчёты. Они служат своеобразным кэшом для быстрых выборок данных.

**Пример:** Компания хочет регулярно видеть отчёт о выручке за месяц. Можно создать специальную таблицу, где ежемесячно накапливаются итоговые показатели, что ускорит формирование нужных отчётов.

### 4. Добавление внешних ключей:

Часто внешними ключами пренебрегают ради уменьшения числа JOIN'ов, но разумное их использование может упростить управление данными и контроль целостности.

**Пример:** Внешний ключ, указывающий на конкретный отдел, можно оставить в таблице сотрудников, чтобы упростить проверку принадлежности сотрудника к отделу.

## Риски денормализации

Несмотря на очевидные плюсы, денормализация таит ряд рисков:

* **Потеря целостности данных.** Поскольку одни и те же данные хранятся в нескольких местах, возможны расхождения, если обновление не произошло синхронно.
* **Рост объёма данных.** Повышается потребность в памяти и дисковом пространстве.
* **Дополнительная нагрузка на сервер.** Поддерживать актуальные копии данных сложнее, что может требовать дополнительного программного кода и временных затрат.

## Когда стоит применять денормализацию?

Денормализацию целесообразно использовать в ситуациях, когда повышение производительности важнее чистоты и отсутствия избыточности данных. Вот несколько случаев, когда она оправдана:

* Для формирования оперативных отчетов и анализа данных.
* В приложениях с большим количеством пользователей и высоким уровнем одновременных обращений.
* При обработке больших объёмов данных в режиме real-time.

## Понятие "Хранилище данных"

## Чем занимается хранилище данных?

Главная задача хранилища данных — объединить разнородные данные из различных информационных систем компании (ERP, CRM, бухгалтерских систем и т.д.) в единый централизованный источник. Затем эти данные очищаются, преобразовываются и организуются в удобном формате для последующего анализа и подготовки отчетности.

Простыми словами, хранилище данных подобно библиотеке, где собираются и классифицируются материалы из разных архивов, чтобы сотрудники могли удобно находить необходимую информацию и извлекать полезные знания.

## Отличительные черты хранилища данных:

1. **Централизованное хранение:** Данные собираются из множества источников и консолидируются в единое пространство.
2. **Интеграция данных:** Информация обрабатывается и приводится к единой структуре, что облегчает дальнейший анализ.
3. **Историчность:** Хранилище сохраняет историю изменений данных, позволяя проводить ретроспективный анализ за прошлые периоды.
4. **Анализ и принятие решений:** Хранилище поддерживает процессы Business Intelligence (BI), предоставляя инструменты для анализа тенденций, выявления закономерностей и составления прогнозов.
5. **Независимость от операционной деятельности:** Хранилище отделено от текущих производственных систем, что гарантирует сохранность данных и независимость аналитических процедур от повседневных рабочих нагрузок.

## **Единый централизованный источник достоверных данных** для всей организации, объединяющий информацию из разных систем в согласованном формате.

## **5 ключевых задач, которые решает DWH**

### **1. Создание "единой версии правды" (Single Source of Truth)**

* Устранение противоречий между отчетами разных отделов
* Пример: Финансовый и продажный отделы видят одинаковые цифры выручки

### **2. Поддержка сложной аналитики**

* Кросс-системный анализ (например, "Как рекламные расходы влияют на продажи и логистику?")
* Многомерный анализ (OLAP): срезы по времени, регионам, категориям

### **3. Исторический анализ**

* Тренды за 5-10 лет
* Сезонность и долгосрочные закономерности
* Пример: "Сравнение продаж новогодних товаров за последние 5 лет"

### **4. Поддержка BI и визуализации**

* Интерактивные дашборды (Power BI, Tableau)
* Автоматизированные отчеты
* Пример: "Ежедневный KPI-дашборд для топ-менеджмента"

### **5. Подготовка данных для Data Science**

* Чистые, структурированные данные для ML-моделей
* Пример: "Прогнозирование спроса с помощью нейросетей"

## **Технические особенности реализации**

1. **ETL/ELT-процессы**:
   * Ежедневное/еженедельное обновление данных
   * Пример инструментов: Informatica, Talend, Apache Airflow
2. **Оптимальные модели данных**:
   * Звезда (для простых отчетов)
   * Снежинка (для сложных нормализованных данных)
   * Data Vault (для гибкой масштабируемости)
3. **Типовой стек технологий**:
   * Хранилище: Snowflake, Teradata, MS SQL Server
   * BI: Power BI, Tableau, QlikView
   * Интеграция: Apache Kafka, Fivetran

## Архитектурные особенности хранилища данных:

Классическая архитектура хранилища данных строится по принципу «звезды» или «снежинки».

* **Архитектура «Звезда»:** Включает центральную таблицу фактов (fact table), окруженную измерениями (dimension tables). Такое построение обеспечивает простоту и быстрое выполнение аналитических запросов.
* **Архитектура «Снежинка»:** Более сложная структура, где измерения разбиты на дополнительные суб-измерения, что повышает степень нормализации и экономит память, но усложняет выполнение запросов.

## Преимущества внедрения хранилища данных:

1. **Единообразие данных:** Сбор данных из разных источников в стандартизированном формате позволяет сравнивать и анализировать информацию независимо от первоначальных различий.
2. **Скорость и удобство анализа:** Предусмотренные механизмы агрегации и индексации обеспечивают быструю подготовку отчетов и поддержку интерактивных аналитических инструментов.
3. **Комплексный анализ бизнеса:** Объединение данных из различных сфер деятельности компании открывает возможности для глубокого изучения взаимосвязей и выявления скрытых факторов успеха.
4. **История изменений:** Наличие исторической хронологии данных позволяет строить прогнозы и оценивать динамику развития компании.

## Важные технологии и подходы:

* **ETL-процессы (Extract, Transform, Load):** Механизмы сбора, преобразования и загрузки данных в хранилище.
* **OLAP-технологии (Online Analytical Processing):** Специальные методы для быстрого выполнения аналитических запросов.
* **Моделирование данных:** Определение метаданных, структуры таблиц и способов интеграции данных.

## Применение хранилищ данных:

Современные крупные компании активно внедряют хранилища данных для нужд оперативного анализа и стратегического планирования. Наиболее востребованными направлениями являются финансовая сфера, розничная торговля, телекоммуникации и производство, где накоплено огромное количество данных, нуждающихся в глубокой аналитике.

Корпоративное хранилище данных (КХД, англ. Enterprise Data Warehouse, EDW или DWH) — это специальная инфраструктура для сбора, хранения и анализа больших объемов данных компании, собранных из различных внутренних и внешних источников. Его главная задача — предоставление единого информационного пространства для аналитиков, менеджеров и специалистов, принимающих управленческие решения.

Вот ключевые задачи, решаемые корпоративным хранилищем данных (DWH):

## 1. Интеграция и унификация данных

Компании часто сталкиваются с проблемой рассредоточенности данных: ERP-системы, CRM, бухгалтерские программы, складские учеты и прочие информационные ресурсы функционируют обособленно. Корпоративное хранилище объединяет все эти разрозненные данные в единую инфраструктуру, что создает основу для комплексного анализа.

**Зачем это нужно?**

* Обеспечение единства и согласованности данных, поступающих из разных источников.
* Исключение дублирования информации и создание универсального представления о бизнесе.

## 2. Историчность и анализ динамики

Хранилище данных сохраняет полную историю изменений данных, что позволяет специалистам смотреть на информацию в динамике. Например, благодаря такому механизму можно оценить, как менялись продажи компании за последние годы, проанализировать тенденции и построить долгосрочные прогнозы.

**Какие выгоды это приносит?**

* Оценка долгосрочных показателей деятельности компании.
* Выявление сезонных колебаний спроса, выявление узких мест и точек роста.

## 3. Анализ и принятие решений

Основное назначение DWH — обеспечение высокой доступности данных для анализа и выработки стратегических решений. Специалисты получают возможность быстро получать ответы на вопросы вроде: «Какой продукт лучше продается?» или «Где сконцентрированы самые прибыльные клиенты?»

**Преимущества такого подхода:**

* Быстрое получение достоверной информации для принятия важных решений.
* Использование аналитических инструментов для выявления новых перспектив и угроз бизнесу.

## 4. Улучшение качества данных

Хранилище данных оснащено механизмами очистки и проверки информации. Прежде чем попасть в хранилище, данные проходят процедуру контроля качества, исключения некорректных записей и заполнения недостающей информации. Такой подход позволяет гарантировать точность и надежность предоставляемых данных.

**Результат:**

* Высокоуровневые отчёты и рекомендации основаны на проверенных и надежных данных.
* Минимизация влияния человеческого фактора и снижение количества ошибок.

## 5. Оптимизация бизнес-процессов

Благодаря KHD/DWH руководители получают представление обо всех аспектах деятельности компании, что позволяет своевременно обнаруживать слабые звенья и оптимизировать рабочие процессы. Например, система может показать, какие логистические маршруты экономически невыгодны или какие маркетинговые кампании оказались наименее успешными.

**Польза для бизнеса:**

* Рационализация расходов и сокращение издержек.
* Увеличение доходов за счёт грамотного распределения ресурсов.

## 6. Комплексный анализ и интеграция с BI-инструментами

Корпоративное хранилище отлично сочетается с инструментами Business Intelligence (BI), такими как Power BI, Tableau или QlikView. Вместе они предоставляют мощные средства для визуализации данных, подготовки отчетов и ведения аналитики.

**Что это даёт?**

* Быстрая подготовка наглядных графиков и диаграмм для понимания текущего положения дел.
* Возможность оперативно реагировать на изменения рынка и конкурентов.

## Итоговая ценность КХД/DWH:

Корпоративное хранилище данных играет важную роль в современном управлении предприятием, решая широкий спектр задач:

* Предоставляет целостную картину деятельности компании.
* Помогает принимать информированные решения на основании объективных данных.
* Способствует развитию конкурентоспособности и улучшению финансовых показателей.

Таким образом, внедрение и эффективное использование корпоративных хранилищ данных становится важным фактором успешного функционирования компаний любого масштаба и отрасли.

## OLAP, OLTP, в чём отличие и почему OLTP не подходит для аналитических систем

# **OLAP vs OLTP**

## **1. Основные понятия**

### **OLTP (Online Transaction Processing)**

* **Что это?** Системы для **операционной обработки транзакций** в реальном времени
* **Главная задача**: Быстрая запись и обновление данных
* **Примеры**:
  + Банковские переводы
  + Оформление заказов в интернет-магазине
  + Запись пациентов в поликлинике

### **OLAP (Online Analytical Processing)**

* **Что это?** Системы для **анализа больших объемов данных**
* **Главная задача**: Быстрое выполнение сложных аналитических запросов
* **Примеры**:
  + Анализ продаж за 5 лет
  + Прогнозирование спроса
  + Финансовая отчетность

## **2. Ключевые различия**

| **Критерий** | **OLTP** | **OLAP** |
| --- | --- | --- |
| **Назначение** | Операционная работа | Аналитика и отчетность |
| **Тип запросов** | Короткие UPDATE/INSERT | Долгие SELECT с GROUP BY |
| **Структура** | Нормализованная (3НФ) | Денормализованная (звезда) |
| **Данные** | Только актуальные | Исторические (годы) |
| **Оптимизация** | Для записи | Для чтения |
| **Размер операций** | Много мелких | Мало крупных |

## **3. Почему OLTP не подходит для аналитики?**

1. **Медленные сложные запросы**
   * OLTP оптимизированы для поиска по ключу (WHERE id = 123), а не для агрегаций (SUM(sales))
2. **Нет исторических данных**
   * В OLTP обычно хранятся только актуальные данные (например, текущий баланс счета)
3. **Высокая нагрузка**
   * Аналитический запрос может "положить" операционную систему
4. **Нет аналитических структур**
   * Отсутствуют предварительные агрегаты и оптимизированные индексы

**Пример проблемы**:  
Попытка построить отчет "Динамика продаж по месяцам за 3 года" в OLTP-системе интернет-магазина приведет к:

* 15-минутному выполнению запроса
* Блокировке таблиц заказов
* Ошибкам при оформлении новых заказов

## **4. Преимущества OLAP**

✔ **Высокая скорость аналитических запросов** (в 100-1000 раз быстрее OLTP)  
✔ **Хранение многолетней истории**  
✔ **Поддержка сложных агрегаций** (ROLLUP, CUBE, оконные функции)  
✔ **Оптимизированная структура** (звезда, снежинка, columnar storage)  
✔ **Не мешает операционной работе** (отдельная система)

**Пример выгоды**:  
Отчет, который в OLTP выполнялся 2 часа, в OLAP делается за 10 секунд

## **5. Преимущества OLTP**

✔ **Мгновенная обработка транзакций** (например, перевод денег за 0.1 сек)  
✔ **Гарантированная целостность данных** (ACID-транзакции)  
✔ **Поддержка тысяч операций в секунду**  
✔ **Минимальные задержки** для пользователей

**Пример выгоды**:  
500 касс магазина могут одновременно пробивать чеки без тормозов

## **OLTP (Online Transaction Processing)**

**Назначение:** Операционная обработка транзакций в реальном времени  
**Характеристики:**

* Частые короткие операции (INSERT/UPDATE/DELETE)
* Нормализованная структура (3НФ/НФБК)
* Акцент на целостность данных

### **Примеры OLTP-систем**

| **Система** | **Пример использования** | **Технологии** |
| --- | --- | --- |
| **Банковская система** | Переводы между счетами, платежи | Oracle, PostgreSQL |
| **Интернет-магазин** | Оформление заказов, списание остатков | MySQL, MS SQL Server |
| **Медицинская запись** | Запись к врачу, обновление карты пациента | Couchbase, MongoDB |
| **CRM (Salesforce)** | Добавление клиента, изменение сделки | PostgreSQL, DynamoDB |
| **Билетная система** | Бронирование авиабилетов | Redis, MariaDB |

**Пример SQL-запроса (OLTP):**

sql

Copy

Download

UPDATE accounts SET balance = balance - 100

WHERE account\_id = '12345';

## **OLAP (Online Analytical Processing)**

**Назначение:** Анализ больших объемов исторических данных  
**Характеристики:**

* Сложные read-only запросы (агрегации, GROUP BY)
* Денормализованная структура (звезда/снежинка)
* Оптимизация для быстрого чтения

### **Примеры OLAP-систем**

| **Система** | **Пример использования** | **Технологии** |
| --- | --- | --- |
| **Финансовая аналитика** | Годовой отчет по прибыли по филиалам | Snowflake, Redshift |
| **Retail DWH** | Анализ продаж за 5 лет по категориям | Teradata, ClickHouse |
| **Маркетинговая аналитика** | ROI рекламных кампаний за квартал | BigQuery, Vertica |
| **Логистическая аналитика** | Оптимизация маршрутов доставки | SAP HANA, Greenplum |
| **Healthcare Analytics** | Анализ эффективности лечения по регионам | Databricks, Exasol |

**Пример 1: Супермаркет**

* **OLTP-система**:

INSERT INTO orders VALUES (1001, '2023-10-25', 5, 149.99); *-- Регистрация покупки*

* **OLAP-система**:

*-- Анализ продаж молочных продуктов по месяцам*

SELECT

EXTRACT(MONTH FROM date) AS month,

SUM(amount) AS milk\_sales

FROM sales

WHERE category = 'Молочные продукты'

GROUP BY month;

## **Когда что использовать?**

* **Выбирайте OLTP**, если:
  + Нужна запись данных в реальном времени (касса, банкомат)
  + Важна актуальность данных "здесь и сейчас"
* **Выбирайте OLAP**, если:
  + Требуется анализ исторических данных
  + Нужны сложные агрегации (SUM, AVG, PIVOT)
  + Приоритет — скорость чтения, а не записи

## Что такое ETL и для чего он нужен. Основные проблемы. Отличие ETL от ELT подхода.

ETL (Extract, Transform, Load) — это важный технологический процесс, предназначенный для извлечения, трансформации и загрузки данных из различных источников в целевую систему, такую как хранилище данных (Data Warehouse) или озеро данных (Data Lake). Давайте поговорим подробно о том, что такое ETL, зачем он нужен, какие проблемы возникают в ходе его использования и чем он отличается от альтернативного подхода ELT.

## Что такое ETL?

ETL расшифровывается как Extract (извлечение), Transform (преобразование) и Load (загрузка). Суть этого процесса заключается в следующем:

1. **Extract (Извлечение):** Сбор данных из различных источников (операционных систем, файлов, баз данных и т.д.).
2. **Transform (Преобразование):** Изменение формата и содержания данных согласно потребностям целевой системы. Сюда входят очистку данных, фильтрацию, конвертацию типов данных, удаление дубликатов и другие манипуляции.
3. **Load (Загрузка):** Перенос преобразованных данных в целевое хранилище или систему для дальнейшей обработки и анализа.

Таким образом, ETL выступает посредником между разнородными системами, собирая и стандартизируя данные для последующих аналитических и бизнес-задач.

## Зачем нужен ETL?

. В компаниях нередко имеются десятки и сотни информационных систем, каждая из которых генерирует собственные данные в собственных форматах. Без механизма переноса и приведения данных к общему виду невозможно полноценно провести аналитику и принять верные управленческие решения.

Именно тут вступает в игру ETL. Его основные цели:

* Собрать всю необходимую информацию из разных источников.
* Привести данные к единому формату, сделав их пригодными для анализа.
* Загрузить трансформированные данные в хранилище, откуда они станут доступны для бизнес-аналитики и принятия решений.

## Проблемы ETL-подхода

Хотя ETL широко используется и полезен, у него есть ряд недостатков и потенциальных трудностей:

1. **Медленная обработка больших объемов данных:** Современные корпорации производят колоссальные объёмы данных, и традиционная процедура ETL (особенно на этапе преобразования) может стать бутылочным горлышком в обработке информации.
2. **Ограниченность масштабирования:** Традиционные ETL-системы испытывают трудности при увеличении объемов данных и числа источников, что негативно сказывается на производительности.
3. **Необходимость постоянной настройки:** Из-за разнообразия источников данных, форматов и типов данных ETL-процесс требует регулярного мониторинга и тонкой настройки, что добавляет сложности в эксплуатацию.
4. **Потери данных:** Во время процедуры трансформации возможна потеря данных или искажение информации, что ставит под угрозу качество аналитики.

## Альтернативный подход — ELT (Extract, Load, Transform)

ELT — это новый подход, возникший как альтернатива традиционным ETL-системам. Он меняет последовательность шагов:

1. **Extract (Извлечение):** Получение сырых данных из различных источников.
2. **Load (Загрузка):** Непосредственное помещение необработанных данных в целевую систему (чаще всего облачную платформу).
3. **Transform (Преобразование):** Выполнение преобразований данных непосредственно в целевой среде (например, облаке), используя мощности целевой инфраструктуры.

### Отличия ETL от ELT:

* **Место трансформации:** В ETL данные сначала трансформируются, потом загружаются. В ELT данные загружаются в сыром виде, а затем преобразуются в целевой системе.
* **Производительность:** ELT лучше справляется с большими объемами данных, так как мощность для трансформации предоставляется целевой платформой (например, облачным сервисом).
* **Эластичность:** ELT более эластичен и легко масштабируется по мере увеличения объемов данных.

## Какой подход выбрать?

Выбор между ETL и ELT зависит от особенностей вашей среды и потребностей бизнеса:

* **ETL предпочтителен:** Если ваши данные сравнительно невелики, они поступают из стабильных источников и не требуют мощных средств обработки.
* **ELT рекомендуется:** Если у вас огромный поток данных, большое число источников и потребность в гибкости и масштабируемости.

### Информация для аттестации: ETL и ELT

#### Что такое ETL и для чего он нужен?

**ETL** (Extract, Transform, Load — Извлечение, Трансформация, Загрузка) — это процесс, используемый в управлении данными для интеграции, очистки и подготовки данных для анализа, отчетности или хранения в хранилищах данных (Data Warehouse).

1. **Извлечение (Extract)**: Данные извлекаются из различных источников, таких как базы данных, API, файлы (CSV, Excel), облачные сервисы и т.д.
2. **Трансформация (Transform)**: Данные преобразуются в нужный формат: очищаются (удаление дубликатов, обработка пропусков), агрегируются, обогащаются, преобразуются в соответствии с бизнес-логикой.
3. **Загрузка (Load)**: Подготовленные данные загружаются в целевую систему, обычно в хранилище данных (например, Snowflake, Redshift) или базу данных для дальнейшего анализа.

**Для чего нужен ETL?**

* **Интеграция данных**: Объединяет данные из разных источников в единую систему.
* **Подготовка к анализу**: Обеспечивает чистоту, структурированность и совместимость данных для аналитики и BI-инструментов.
* **Автоматизация процессов**: Упрощает регулярную обработку больших объемов данных.
* **Поддержка бизнес-решений**: Позволяет компаниям принимать обоснованные решения на основе качественных данных.

**Примеры использования**:

* Создание отчетов для BI-платформ (Tableau, Power BI).
* Формирование хранилищ данных для аналитики.
* Миграция данных между системами.

#### Основные проблемы ETL

1. **Сложность источников данных**:
   * Разнородные форматы и структуры данных (реляционные базы, JSON, NoSQL).
   * Проблемы с доступом к источникам (разные API, ограничения безопасности).
2. **Качество данных**:
   * Пропущенные значения, дубликаты, несоответствия форматов.
   * Ошибки в данных, требующие сложной очистки.
3. **Производительность**:
   * Обработка больших объемов данных может быть медленной.
   * Ограничения по вычислительным ресурсам или пропускной способности сети.
4. **Сложность трансформаций**:
   * Сложные бизнес-правила требуют глубокого понимания данных.
   * Изменение требований может привести к переработке ETL-процессов.
5. **Сопровождение и масштабируемость**:
   * Поддержка ETL-пайплайнов требует значительных ресурсов.
   * Сложность адаптации процессов при добавлении новых источников данных.
6. **Задержки в обработке**:
   * В реальном времени ETL может быть сложным из-за необходимости быстрой трансформации.
7. **Безопасность и соответствие требованиям**:
   * Обеспечение защиты данных (GDPR, HIPAA).
   * Управление доступом к чувствительным данным.

#### Отличие ETL от ELT

**ELT** (Extract, Load, Transform — Извлечение, Загрузка, Трансформация) — это подход, при котором данные сначала загружаются в целевую систему, а затем трансформируются.

| **Критерий** | **ETL** | **ELT** |
| --- | --- | --- |
| **Порядок этапов** | Извлечение → Трансформация → Загрузка | Извлечение → Загрузка → Трансформация |
| **Место трансформации** | Трансформация выполняется до загрузки в хранилище (на сервере ETL). | Трансформация выполняется в целевой системе (обычно в облачном DWH). |
| **Скорость** | Может быть медленнее из-за предварительной трансформации. | Быстрее, так как данные загружаются в сыром виде, а трансформация — позже. |
| **Гибкость** | Менее гибкий, требует заранее определенных правил трансформации. | Более гибкий, подходит для работы с сырыми данными и Ad-Hoc анализа. |
| **Технологии** | Традиционные ETL-инструменты (Informatica, Talend, SSIS). | Облачные платформы (Snowflake, BigQuery, Databricks). |
| **Объем данных** | Лучше для меньших объемов, где трансформация заранее оптимизирована. | Подходит для больших объемов данных (Big Data). |
| **Использование** | Традиционные хранилища данных, строгие схемы. | Облачные хранилища, где вычислительная мощность доступна для трансформации. |

**Когда использовать ETL?**

* Данные требуют сложной предварительной обработки перед загрузкой.
* Целевая система имеет ограниченные вычислительные ресурсы.
* Необходима строгая структура данных (например, для реляционных баз данных).

**Когда использовать ELT?**

* Работа с большими объемами данных (Big Data).
* Использование мощных облачных хранилищ, где трансформация выполняется быстрее.
* Нужна гибкость для аналитиков (Ad-Hoc запросы, эксперименты с данными).

## Dimension model. Какие данные должны хранится в Dimension.

## Что такое дименшн (Dimension)?

Дименшн (измерение) — это совокупность данных, описывающих определенный контекст для факта (факта данных). Измерения добавляют семантику и глубину пониманию цифр, содержащихся в фактах. Они помогают интерпретировать и осмыслять информацию, находящуюся в таблицах фактов, и предоставляют дополнительную информацию о событиях, объектах или процессах, зафиксированных в этих данных.

### Ключевые характеристики измерений:

* Измерения отвечают на вопросы «когда», «где», «почему», «кем» и «чем», давая возможность глубже изучить данные.
* Измерения обычно состоят из относительно небольшого количества уникальных значений, но могут содержать большое количество атрибутов.
* Они образуют "контекст", в котором находятся фактические данные, и позволяют представлять информацию в удобной для анализа форме.

### Примеры измерений:

* Временные измерения (календарь, дни, месяцы, годы)
* Географическое измерение (города, регионы, страны)
* Измерение товаров (названия, бренды, категории)
* Измерение покупателей (демографические данные, предпочтения)

## Как определить, что выделить в измерение?

Определить, какие столбцы выделить в измерение, можно следуя ряду критериев:

1. **Частота использования в аналитических запросах:**Если атрибут регулярно запрашивается в отчетах и анализах, он заслуживает выделения в отдельное измерение.
2. **Семантическая важность:**Данные, которые несут смысловую нагрузку и влияют на восприятие фактов, выделяются в измерения.
3. **Возможность группировки и детализации:**Атрибуты, которые позволяют сегментировать данные и посмотреть их в различных разрезах, подходят для включения в измерения.
4. **Постоянство значений:**Если значения атрибута меняются редко или вообще постоянны, его лучше поместить в измерение, так как оно станет стабильной основой для анализа.

## Пример: выделение измерений из таблицы Sell\_Out

Допустим, у нас есть таблица Sell\_Out (продажи), содержащая данные о продаже товаров магазина. Она может выглядеть примерно так:

| SaleID | ProductCode | Category | StoreID | Region | SalesDate | QuantitySold | Price |

|--------|-------------|----------|---------|--------|-----------|--------------|-------|

| 1 | ABC123 | Electronics| STO001 | Moscow | 2023-01-01| 5 | 1000 |

| 2 | DEF456 | Clothing | STO002 | SPb | 2023-01-02| 10 | 500 |

Посмотрим, какие столбцы можно выделить в измерения:

1. **Time dimension (временное измерение):**
   * SalesDate: Отдельное измерение времени даст возможность анализировать продажи по дням, месяцам, сезонам и другим периодам.
2. **Product dimension (измерение товаров):**
   * ProductCode, Category: Выделяются в отдельное измерение товаров, чтобы детально анализировать ассортимент и группы товаров.
3. **Store dimension (измерение магазинов):**
   * StoreID, Region: Создается измерение для анализа активности продаж по регионам и магазинам.

Таким образом, в результате выделения измерений таблица примет следующий вид:

* **Фактическая таблица (Fact table) Sell\_Out:**
* | SaleID | ProductKey | StoreKey | TimeKey | QuantitySold | Price |
* |--------|------------|----------|---------|--------------|-------|
* | 1 | PRD001 | STR001 | TM001 | 5 | 1000 |

| 2 | PRD002 | STR002 | TM002 | 10 | 500 |

* **Измерения:**
  + **Products dimension:**
  + | ProductKey | ProductCode | Category |
  + |------------|-------------|----------|
  + | PRD001 | ABC123 | Electronics|

| PRD002 | DEF456 | Clothing |

* + **Stores dimension:**
  + | StoreKey | StoreID | Region |
  + |----------|---------|--------|
  + | STR001 | STO001 | Moscow |

| STR002 | STO002 | SPb |

* + **Time dimension:**
  + | TimeKey | SalesDate | Year | Month | Day |
  + |---------|-----------|------|-------|-----|
  + | TM001 | 2023-01-01| 2023 | Jan | 1 |

| TM002 | 2023-01-02| 2023 | Jan | 2 |

Такая структура значительно облегчает последующий анализ и позволяет аналитикам оперировать большими объемами данных гораздо эффективнее.

* **Факты (Fact Tables)**: Содержат количественные данные (метрики), такие как суммы продаж, количество заказов, выручка.
* **Измерения (Dimension Tables)**: Содержат описательные атрибуты, которые дают контекст для фактов (например, кто, где, когда, что).

Модель измерений упрощает сложные аналитические запросы, делая данные интуитивно понятными для аналитиков и BI-инструментов.

#### Какие данные должны храниться в Dimension?

**Измерения (Dimensions)** — это таблицы, которые содержат **описательные** и **контекстные** данные, используемые для фильтрации, группировки и анализа фактов. Они обычно представляют сущности, которые описывают бизнес-процессы, такие как клиенты, продукты, время, география и т.д.

**Типы данных, хранящихся в Dimension**:

1. **Идентификаторы**:
   * Уникальный ключ измерения (Surrogate Key), который связывает таблицу измерений с таблицей фактов. Например, CustomerID, ProductID.
   * Используется для обеспечения уникальности и производительности запросов.
2. **Описательные атрибуты**:
   * Атрибуты, которые описывают сущность. Например:
     + Для измерения **Клиент**: имя, фамилия, email, возраст, пол, город проживания.
     + Для измерения **Продукт**: название, категория, бренд, цена, цвет.
     + Для измерения **Время**: дата, день недели, месяц, квартал, год.
   * Эти атрибуты используются для фильтрации и группировки в аналитических запросах.
3. **Иерархии**:
   * Структурированные данные, которые позволяют выполнять анализ на разных уровнях детализации. Например:
     + Для измерения **География**: страна → регион → город → почтовый индекс.
     + Для измерения **Время**: год → квартал → месяц → день.
   * Иерархии упрощают выполнение drill-down (углубление) и roll-up (агрегация) запросов.
4. **Метаданные и категории**:
   * Дополнительные атрибуты, которые классифицируют или обогащают данные. Например:
     + Для измерения **Продукт**: статус (активный/снят с производства), тип упаковки.
     + Для измерения **Клиент**: сегмент (VIP, обычный), дата регистрации.
5. **Атрибуты для фильтрации и анализа**:
   * Логические или категориальные поля, которые часто используются в отчетах. Например:
     + Для измерения **Время**: является ли день выходным (да/нет), сезон (зима/лето).
     + Для измерения **Клиент**: уровень лояльности (высокий/низкий).
6. **Исторические данные (Slowly Changing Dimensions, SCD)**:
   * Для учета изменений данных с течением времени. Например:
     + **SCD Type 1**: Перезапись старых данных (например, обновление адреса клиента).
     + **SCD Type 2**: Сохранение истории изменений (добавление новой записи с датой действия).
     + **SCD Type 3**: Хранение ограниченной истории (например, текущий и предыдущий адрес).
   * Пример: Для измерения **Клиент** хранится история изменения адреса с датами действия.

#### Основные характеристики данных в Dimension

1. **Денормализация**:
   * Измерения обычно денормализованы (в отличие от снежинковой схемы) для упрощения запросов и повышения производительности.
   * Например, в таблице **География** хранятся все уровни иерархии (страна, регион, город) в одной таблице.
2. **Статичность или медленное изменение**:
   * Данные в измерениях обычно стабильны или медленно меняются (например, клиент редко меняет имя или город).
3. **Малый объем данных**:
   * По сравнению с таблицами фактов, таблицы измерений содержат меньше строк, но больше описательных столбцов.
4. **Уникальность**:
   * Каждая запись в измерении имеет уникальный ключ (Surrogate Key), который используется для связи с таблицами фактов.

#### Основные проблемы при работе с Dimension

1. **Управление изменениями (SCD)**:
   * Реализация Slowly Changing Dimensions может быть сложной, особенно для Type 2, где нужно хранить историю.
   * Требует дополнительных полей (например, StartDate, EndDate, IsActive).
2. **Дублирование данных**:
   * При денормализации могут возникать избыточные данные, увеличивающие объем хранилища.
3. **Согласованность данных**:
   * Необходимо обеспечить консистентность данных между источниками и измерениями.
4. **Производительность**:
   * Слишком большие таблицы измерений могут замедлять запросы, если не оптимизированы.

## Fact, какие данные должны хранится в Fact.

### Fact Tables и данные, которые должны в них храниться

#### Что такое Fact Table?

**Таблица фактов (Fact Table)** — это центральный элемент модели данных в хранилищах данных (Data Warehouse), используемый в звездообразной (Star Schema) или снежинковой (Snowflake Schema) схеме. Fact Table содержит **количественные** или **измеримые** данные (метрики), которые представляют бизнес-показатели, а также ссылки на таблицы измерений (Dimension Tables), которые предоставляют контекст для этих метрик.

**Назначение Fact Table**:

* Хранить числовые данные, которые анализируются для принятия бизнес-решений (например, суммы продаж, количество заказов, доход).
* Служить основой для аналитических запросов, таких как агрегация, фильтрация и группировка.
* Связывать метрики с измерениями (например, кто, где, когда) через внешние ключи.

**Примеры Fact Tables**:

* Таблица продаж: суммы продаж, количество проданных единиц.
* Таблица транзакций: суммы транзакций, время транзакции.
* Таблица инвентаризации: уровень запасов, количество перемещений товаров.

#### Какие данные должны храниться в Fact Table?

**Fact Table** содержит следующие типы данных:

1. **Метрики (числовые показатели)**:
   * Это основные количественные данные, которые подлежат анализу. Они могут быть:
     + **Аддитивные**: Суммируются по любому измерению (например, сумма продаж, количество проданных единиц).
     + **Полуаддитивные**: Суммируются только по определенным измерениям (например, баланс счета на конец дня — суммируется по клиентам, но не по времени).
     + **Неаддитивные**: Не подлежат суммированию (например, процент скидки, коэффициент).
   * Примеры: SalesAmount, OrderQuantity, Revenue, Profit, DiscountPercentage.
2. **Внешние ключи (Foreign Keys)**:
   * Ссылки на первичные ключи таблиц измерений (Dimension Tables), которые обеспечивают контекст для метрик.
   * Например: CustomerID, ProductID, DateID, StoreID.
   * Эти ключи связывают Fact Table с Dimension Tables, позволяя фильтровать и группировать данные.
3. **Метаданные (опционально)**:
   * Дополнительные атрибуты, которые помогают в анализе, но не являются основными метриками. Например:
     + Тип транзакции (наличные/карта).
     + Код валюты.
   * Эти данные обычно минимальны, так как основная описательная информация хранится в таблицах измерений.
4. **Временные метки (опционально)**:
   * Иногда Fact Table содержит временные метки (например, точное время транзакции), если требуется более детализированный анализ, чем предоставляет измерение времени.

**Характеристики данных в Fact Table**:

* **Высокий объем строк**: Fact Table обычно содержит значительно больше записей, чем таблицы измерений, так как фиксирует каждое событие (например, каждую продажу).
* **Нормализация**: Fact Table нормализована, чтобы минимизировать избыточность и обеспечить производительность.
* **Числовой характер**: Большинство столбцов — это числовые данные для агрегации.

#### Как определить Fact Table в таблице?

Чтобы определить, является ли таблица Fact Table, нужно проверить следующие критерии:

1. **Содержит количественные метрики**:
   * Таблица должна включать числовые показатели, которые можно агрегировать (например, сумма, среднее, количество).
   * Пример: Если таблица содержит SalesAmount или OrderQuantity, это, скорее всего, Fact Table.
2. **Содержит внешние ключи к измерениям**:
   * Таблица должна иметь столбцы, которые ссылаются на первичные ключи таблиц измерений (например, CustomerID, DateID).
   * Это отличает Fact Table от Dimension Table, которая содержит описательные данные.
3. **Представляет бизнес-события**:
   * Fact Table фиксирует события или транзакции, такие как продажи, заказы, доставки, клики и т.д.
   * Пример: Таблица с записями о каждой продаже в магазине — это Fact Table.
4. **Поддерживает аналитические операции**:
   * Данные в таблице используются для выполнения операций SUM, COUNT, AVG, MIN, MAX и других агрегатов.
   * Если таблица используется для группировки и фильтрации по измерениям, это Fact Table.

**Пример структуры Fact Table**: **Таблица фактов "Продажи"**:

| **SaleID** | **CustomerID** | **ProductID** | **DateID** | **StoreID** | **SalesAmount** | **Quantity** | **Discount** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 101 | 201 | 301 | 401 | 150.50 | 2 | 10% |
| 2 | 102 | 202 | 302 | 401 | 89.99 | 1 | 5% |

* **SaleID**: Уникальный идентификатор транзакции (опционально, первичный ключ).
* **CustomerID**, **ProductID**, **DateID**, **StoreID**: Внешние ключи, ссылающиеся на таблицы измерений (Клиент, Продукт, Время, Магазин).
* **SalesAmount**, **Quantity**, **Discount**: Метрики для анализа.

#### Типы Fact Tables

1. **Transactional Fact Table**:
   * Хранит данные о конкретных событиях (например, каждая продажа, транзакция).
   * Пример: Таблица продаж с записями о каждой покупке.
2. **Snapshot Fact Table**:
   * Хранит данные на определенный момент времени (например, остатки на складе на конец дня).
   * Пример: Таблица с ежедневными остатками банковских счетов.
3. **Accumulating Snapshot Fact Table**:
   * Отслеживает процесс с несколькими стадиями (например, жизненный цикл заказа: размещен, отправлен, доставлен).
   * Пример: Таблица с датами этапов обработки заказа.

#### Основные проблемы при работе с Fact Tables

1. **Объем данных**:
   * Fact Tables могут содержать миллионы или миллиарды строк, что требует оптимизации хранения и индексации.
2. **Производительность запросов**:
   * Сложные аналитические запросы могут быть медленными без правильной индексации или партиционирования.
3. **Согласованность с измерениями**:
   * Внешние ключи должны быть согласованы с таблицами измерений, чтобы избежать ошибок (например, отсутствующий CustomerID).
4. **Обработка дубликатов**:
   * Необходимо предотвращать дублирование транзакций при загрузке данных.
5. **Исторические изменения**:
   * Если измерения используют Slowly Changing Dimensions (SCD), Fact Table должна корректно ссылаться на актуальные версии измерений.

#### Отличие Fact Table от Dimension Table

| **Критерий** | **Fact Table** | **Dimension Table** |
| --- | --- | --- |
| **Содержимое** | Количественные метрики (суммы, количества). | Описательные атрибуты (имя, город, дата). |
| **Объем данных** | Большое количество строк (транзакции, события). | Меньше строк, описывающих сущности. |
| **Структура** | Нормализована, содержит внешние ключи. | Денормализована, содержит иерархии и описания. |
| **Использование** | Для агрегации и анализа (SUM, AVG). | Для фильтрации и группировки (WHERE, GROUP BY). |
| **Пример** | Продажи: SalesAmount, Quantity. | Клиент: Name, City, Segment. |

## Меры (measures), типы мер.

### "Меры (measures), типы мер: агрегируемые (аддитивные), не агрегируемые, полу-агрегируемые"

#### 1. **Меры (measures)**

Меры — это количественные показатели, используемые для оценки, анализа и описания данных в системах аналитики, базах данных или хранилищах данных. Они представляют числовые значения, которые можно измерить, подсчитать или агрегировать для получения аналитических выводов. Меры обычно используются в контексте многомерного анализа (например, в OLAP-кубах) и связаны с определёнными бизнес-процессами.

Примеры мер:

* Выручка (сумма продаж).
* Количество проданных товаров.
* Средний чек.
* Температура.

#### 2. **Типы мер**

Меры классифицируются по возможности их агрегации (объединения) в зависимости от способа обработки данных. Выделяют три основных типа: агрегируемые (аддитивные), не агрегируемые и полу-агрегируемые.

##### 2.1. **Агрегируемые (аддитивные) меры**

Агрегируемые меры — это меры, которые можно суммировать (или выполнять другие аддитивные операции, такие как сложение) по любым измерениям (например, время, регион, продукт). Такие меры сохраняют корректность при объединении данных.

**Характеристики:**

* Поддерживают арифметические операции (сложение, вычитание).
* Результат агрегации имеет смысл и является правильным.
* Используются для подсчёта итогов, например, в отчётах.

**Примеры:**

* **Выручка**: Сумма продаж за месяц, год или по регионам может быть сложена для получения общей выручки. Например, выручка магазина А (100,000 руб.) + выручка магазина Б (150,000 руб.) = 250,000 руб.
* **Количество проданных единиц товара**: Если в январе продано 50 единиц, а в феврале — 30, то общее количество = 80 единиц.
* **Вес товаров**: Вес товаров в разных партиях можно суммировать для получения общего веса.

##### 2.2. **Не агрегируемые меры**

Не агрегируемые меры — это показатели, которые нельзя суммировать или агрегировать, так как результат не имеет смысла или теряет свою интерпретацию. Такие меры обычно представляют собой характеристики, которые не подлежат сложению.

**Характеристики:**

* Агрегация (например, суммирование) приводит к бессмысленным результатам.
* Часто используются для описания качественных характеристик или уникальных значений.
* Вместо суммирования могут использоваться другие операции, такие как подсчёт уникальных значений или выбор максимума/минимума.

**Примеры:**

* **Температура**: Суммирование температуры за день (например, 20°C утром + 25°C днём) не имеет смысла. Вместо этого можно вычислить среднюю температуру или максимум/минимум.
* **Коды клиентов (ID)**: Суммирование идентификаторов клиентов (например, ID 1001 + ID 1002) бессмысленно.
* **Процент скидки**: Сложение процентов скидки по разным товарам не даёт полезной информации.

##### 2.3. **Полу-агрегируемые меры**

Полу-агрегируемые меры — это меры, которые можно агрегировать по некоторым измерениям, но не по всем. Их агрегация возможна только в определённых контекстах, а в других случаях она теряет смысл.

**Характеристики:**

* Могут суммироваться по одним измерениям (например, по времени), но не по другим (например, по категориям).
* Требуют специальных правил агрегации (например, среднее значение или подсчёт).
* Часто связаны с относительными показателями или производными мерами.

**Примеры:**

* **Средний чек**: Можно суммировать общую выручку и количество транзакций по времени (например, за месяц), а затем вычислить средний чек (выручка / количество транзакций). Однако суммирование среднего чека по разным магазинам напрямую не имеет смысла, так как нужно учитывать вес каждой транзакции.
* **Процент выполнения плана**: Можно агрегировать по времени (например, средний процент выполнения за год), но суммирование по разным регионам требует взвешивания.
* **Запасы на складе**: Можно суммировать запасы по времени или по продуктам, но агрегация по разным категориям товаров может быть некорректной без учёта их специфики.

#### 3. **Примеры в контексте**

Для наглядности рассмотрим пример из розничной торговли:

| **Мера** | **Тип меры** | **Пример использования** |
| --- | --- | --- |
| Выручка | Агрегируемая | Суммирование выручки по всем магазинам за год: 100,000 + 150,000 = 250,000 руб. |
| Средняя температура | Не агрегируемая | Температура в магазине (20°C, 22°C) не суммируется, но можно вычислить среднее: 21°C. |
| Средний чек | Полу-агрегируемая | Средний чек по дням можно вычислить, но прямое суммирование чеков по магазинам некорректно. |

## Гранулярность данных.

### Гранулярность данных для аттестации

#### 1. **Что такое гранулярность данных?**

Гранулярность данных — это уровень детализации или степени разбиения данных в информационной системе, хранилище данных или аналитической модели. Она определяет, насколько мелкими или крупными являются единицы данных, с которыми работает система. Гранулярность влияет на точность анализа, объём хранимых данных и производительность запросов.

**Ключевые аспекты:**

* **Высокая гранулярность** (мелкая детализация): Данные разбиты на более мелкие единицы, что обеспечивает высокую точность, но увеличивает объём данных и сложность обработки.
* **Низкая гранулярность** (грубая детализация): Данные агрегированы на более высоком уровне, что уменьшает объём данных и упрощает обработку, но снижает детализацию.

#### 2. **Примеры гранулярности**

* **Высокая гранулярность**:
  + Продажи магазина с детализацией по каждой транзакции (например, данные о каждой покупке: товар, время, сумма, клиент).
  + Температура, измеряемая каждую минуту в течение дня.
  + Логи активности пользователей на сайте с указанием каждого клика.
* **Низкая гранулярность**:
  + Итоговая выручка магазина за месяц.
  + Средняя дневная температура за месяц.
  + Общее количество посещений сайта за неделю.

#### 3. **Типы гранулярности**

Гранулярность может быть определена по разным измерениям:

* **Временная гранулярность**: Данные собираются по секундам, минутам, часам, дням, месяцам и т.д. Например:
  + Высокая: продажи по часам.
  + Низкая: продажи по годам.
* **Пространственная гранулярность**: Данные разбиты по географическим уровням. Например:
  + Высокая: продажи по каждому магазину в городе.
  + Низкая: продажи по стране.
* **Объектная гранулярность**: Уровень детализации объектов. Например:
  + Высокая: данные по каждому товару в чеке.
  + Низкая: данные по категориям товаров.

#### 4. **Влияние гранулярности**

* **На анализ**: Высокая гранулярность позволяет проводить детальный анализ (например, выявлять тренды по часам), но требует больше ресурсов. Низкая гранулярность удобна для обобщённых выводов, но теряет детали.
* **На хранение**: Высокая гранулярность увеличивает объём данных, требуя больше места и мощностей для хранения.
* **На производительность**: Запросы к данным с высокой гранулярностью выполняются дольше, чем к агрегированным данным.
* **На бизнес-решения**: Выбор гранулярности зависит от целей. Например, для анализа поведения клиентов нужна высокая гранулярность, а для годового отчёта — низкая.

#### 5. **Связь с мерами**

Гранулярность данных тесно связана с типами мер (агрегируемые, не агрегируемые, полу-агрегируемые):

* **Агрегируемые меры**: Легко обрабатываются на любом уровне гранулярности. Например, выручку можно суммировать от транзакций (высокая гранулярность) до годовых итогов (низкая гранулярность).
* **Не агрегируемые меры**: Требуют осторожного выбора гранулярности. Например, температура на уровне минут (высокая гранулярность) может быть усреднена для дня, но суммирование бессмысленно.
* **Полу-агрегируемые меры**: Могут агрегироваться на определённой гранулярности. Например, средний чек корректен для агрегации по дням, но требует пересчёта при изменении гранулярности (например, по магазинам).

#### 6. **Пример в контексте**

Рассмотрим продажи в розничной сети:

* **Высокая гранулярность**: Данные о каждой транзакции (дата, время, магазин, товар, сумма). Позволяет анализировать поведение клиентов, но требует больших ресурсов.
* **Средняя гранулярность**: Данные о продажах по дням для каждого магазина. Подходит для анализа дневных трендов.
* **Низкая гранулярность**: Итоговая выручка сети за месяц. Удобно для стратегического планирования, но теряет детали о конкретных товарах или клиентах.

#### 7. **Как выбирать гранулярность?**

* Определить цель анализа: для детального анализа нужна высокая гранулярность, для обобщённых отчётов — низкая.
* Учитывать ресурсы: высокая гранулярность требует больше места и вычислительных мощностей.
* Балансировать точность и производительность: например, для оперативной аналитики можно агрегировать данные до уровня дней или недель.
* Учитывать тип меры: агрегируемые меры проще адаптировать к разным уровням гранулярности, чем не агрегируемые.

#### 8. **Заключение**

Гранулярность данных определяет, насколько детализированным будет анализ, и напрямую влияет на эффективность работы с данными. Правильный выбор гранулярности позволяет сбалансировать точность, производительность и объём хранимых данных. Для агрегируемых мер гранулярность менее критична, для не агрегируемых и полу-агрегируемых требуется учитывать контекст и методы обработки.

## Что такое суррогатные ключи, для чего нужны и какие приемущества они приносят в DWH.

### Суррогатные ключи в хранилищах данных (DWH): определение, назначение, преимущества, создание и связь с натуральными ключами

#### 1. **Что такое суррогатные ключи?**

Суррогатный ключ — это искусственно созданный уникальный идентификатор, используемый в хранилищах данных (DWH) для обозначения записей в таблицах. Он не имеет бизнес-значения и создаётся исключительно для технических целей, чтобы обеспечить уникальность и упростить управление данными.

**Характеристики:**

* Обычно представлен в виде простого числового значения (например, автоинкрементный идентификатор: 1, 2, 3…).
* Не содержит бизнес-логики или семантической информации (в отличие от натуральных ключей).
* Уникален в пределах таблицы.

#### 2. **Для чего нужны суррогатные ключи в DWH?**

Суррогатные ключи играют ключевую роль в хранилищах данных, решая задачи, связанные с производительностью, управлением данными и целостностью. Основные цели их использования:

* **Обеспечение уникальности**: Гарантируют уникальную идентификацию каждой записи в таблице, даже если бизнес-данные содержат дубликаты или изменения.
* **Упрощение связей**: Используются как первичные и внешние ключи для связи таблиц фактов и измерений в DWH.
* **Обработка изменений данных**: Поддерживают отслеживание изменений в данных (например, в моделях Slowly Changing Dimensions, SCD), сохраняя историю изменений без нарушения целостности.
* **Независимость от бизнес-логики**: Защищают структуру DWH от изменений в бизнес-ключах, которые могут меняться из-за новых требований или источников данных.
* **Оптимизация производительности**: Упрощают индексацию и обработку запросов, так как числовые ключи обрабатываются быстрее, чем сложные или составные бизнес-ключи.

#### 3. **Преимущества суррогатных ключей в DWH**

* **Стабильность**: Суррогатные ключи не зависят от изменений в бизнес-данных (например, если клиент меняет ID или название компании).
* **Производительность**: Числовые ключи (обычно целые числа) занимают меньше места и быстрее обрабатываются в запросах и индексах по сравнению с текстовыми или составными ключами.
* **Универсальность**: Подходят для интеграции данных из разных источников, где бизнес-ключи могут быть несовместимыми или содержать дубликаты.
* **Поддержка SCD**: Упрощают реализацию моделей медленно меняющихся измерений (SCD Type 2, Type 3), позволяя хранить историю изменений без конфликтов.
* **Упрощение ETL-процессов**: Упрощают процессы извлечения, трансформации и загрузки данных (ETL), так как суррогатные ключи создаются централизованно и не зависят от исходных данных.
* **Устранение неоднозначности**: Решают проблему, когда бизнес-ключи из разных источников совпадают, но относятся к разным сущностям.

#### 4. **Откуда берутся суррогатные ключи?**

Суррогатные ключи создаются в процессе проектирования и наполнения хранилища данных, обычно в рамках ETL-процессов. Основные способы их генерации:

* **Автоинкрементные значения**: СУБД (например, SQL Server, PostgreSQL) автоматически присваивает уникальный числовой идентификатор каждой новой записи (например, IDENTITY в SQL Server или SERIAL в PostgreSQL).
* **Генерация в ETL-инструментах**: Инструменты ETL (например, Informatica, Talend, Apache NiFi) создают суррогатные ключи с использованием встроенных функций или алгоритмов (например, последовательности или хэш-функций).
* **Централизованные генераторы ключей**: В сложных DWH может использоваться отдельная система для генерации уникальных ключей, чтобы избежать дублирования при интеграции данных из разных источников.
* **UUID/GUID**: В некоторых случаях используются глобально уникальные идентификаторы, хотя они менее распространены из-за большего размера и меньшей производительности по сравнению с целочисленными ключами.

#### 5. **Как связаны суррогатные и натуральные (бизнес) ключи?**

**Натуральный (бизнес) ключ** — это идентификатор, который имеет бизнес-значение и используется в исходных системах (OLTP) для идентификации сущностей. Например:

* Код клиента (CustomerID: "CUST123").
* Номер заказа (OrderID: "ORD456").
* Артикул товара (SKU: "ABC-789").

**Тип связи между суррогатным и натуральным ключами:**

* **Один-к-одному (в большинстве случаев)**: В таблице измерений (dimension table) каждой записи с натуральным ключом соответствует уникальный суррогатный ключ. Например:
  + Таблица клиентов: Натуральный ключ "CUST123" → Суррогатный ключ "1".
* **Один-ко-многим (в SCD Type 2)**: При использовании модели медленно меняющихся измерений типа 2 (SCD Type 2) один натуральный ключ может быть связан с несколькими суррогатными ключами для отображения исторических изменений. Например:
  + Клиент "CUST123" сменил адрес. В таблице измерений создаются две записи с разными суррогатными ключами (например, "1" и "2"), но одним натуральным ключом ("CUST123").
* **Хранение связи**: В DWH натуральные ключи обычно хранятся в таблицах измерений как атрибуты, а суррогатные ключи используются для связывания таблиц фактов (fact tables) и измерений. Например:
  + Таблица измерений "Клиенты": {SurrogateKey: 1, CustomerID: "CUST123", Name: "Иван", Address: "Москва"}.
  + Таблица фактов "Продажи": {SaleID: 100, CustomerSurrogateKey: 1, Amount: 5000}.

#### 6. **Пример в контексте DWH**

Рассмотрим хранилище данных для розничной сети:

* **Исходные данные (OLTP)**: В системе продаж клиенты идентифицируются по натуральному ключу CustomerID (например, "CUST123").
* **В DWH**:
  + В таблице измерений "Клиенты" создаётся суррогатный ключ CustomerSK (например, 1, 2, 3…).
  + Натуральный ключ CustomerID сохраняется как атрибут: {CustomerSK: 1, CustomerID: "CUST123", Name: "Иван"}.
  + В таблице фактов "Продажи" используется CustomerSK для связи с таблицей измерений, а не CustomerID, что упрощает запросы и защищает от изменений в CustomerID.
* **SCD Type 2**: Если клиент меняет адрес, создаётся новая запись с новым CustomerSK, но тем же CustomerID, чтобы сохранить историю изменений.

#### 7. **Заключение**

Суррогатные ключи — это важный элемент архитектуры DWH, обеспечивающий уникальность, стабильность и производительность. Они создаются автоматически в процессе ETL или с помощью СУБД и заменяют натуральные ключи в связях между таблицами. Их использование упрощает интеграцию данных, поддерживает историчность и повышает эффективность запросов. Связь с натуральными ключами обычно реализуется через таблицы измерений, где натуральные ключи хранятся как атрибуты, а суррогатные ключи используются для связывания с таблицами фактов.

## Принцип построения звезды и снежинки, различия и недостатки.

### Принципы построения схем "Звезда" и "Снежинка" в хранилищах данных (DWH), их различия и недостатки

#### 1. **Принципы построения схемы "Звезда" (Star Schema)**

Схема "Звезда" — это одна из самых популярных моделей для проектирования хранилищ данных, используемая для организации данных в многомерных структурах, таких как OLAP-кубы. Она получила название из-за своей структуры, напоминающей звезду, где в центре находится таблица фактов, окружённая таблицами измерений.

**Структура:**

* **Таблица фактов (Fact Table)**: Содержит количественные меры (например, выручка, количество продаж) и ссылки (внешние ключи) на таблицы измерений. Факты обычно имеют высокую гранулярность (например, данные о каждой транзакции).
* **Таблицы измерений (Dimension Tables)**: Содержат описательные атрибуты, которые используются для фильтрации, группировки и анализа фактов (например, таблица "Клиенты" с атрибутами: ID, имя, город; таблица "Время" с атрибутами: дата, месяц, год).
* **Связи**: Таблица фактов связана с таблицами измерений через суррогатные ключи (или натуральные, если они используются). Каждая таблица измерений напрямую соединена с таблицей фактов, без промежуточных таблиц.

**Пример:**

* Таблица фактов "Продажи": {SaleID, CustomerSK, ProductSK, TimeSK, Amount, Quantity}.
* Таблицы измерений:
  + "Клиенты": {CustomerSK, CustomerID, Name, City}.
  + "Продукты": {ProductSK, ProductID, Name, Category}.
  + "Время": {TimeSK, Date, Month, Year}.

**Особенности:**

* Простая структура с минимальным количеством таблиц.
* Денормализованные таблицы измерений (все атрибуты хранятся в одной таблице, без разбиения на подтаблицы).
* Оптимизирована для запросов аналитики (например, в BI-инструментах).

#### 2. **Принципы построения схемы "Снежинка" (Snowflake Schema)**

Схема "Снежинка" — это более сложная и нормализованная версия схемы "Звезда". Она также состоит из таблицы фактов и таблиц измерений, но таблицы измерений нормализуются, то есть разбиваются на дополнительные подтаблицы, что делает структуру похожей на снежинку.

**Структура:**

* **Таблица фактов**: Аналогична схеме "Звезда", содержит меры и ссылки на таблицы измерений.
* **Таблицы измерений**: Нормализованы, то есть разбиты на несколько связанных таблиц для устранения избыточности данных. Например, таблица "Продукты" может быть разделена на таблицы "Продукты", "Категории" и "Бренды".
* **Связи**: Таблица фактов связана с таблицами измерений, а таблицы измерений могут быть связаны между собой (иерархические связи).

**Пример:**

* Таблица фактов "Продажи": {SaleID, CustomerSK, ProductSK, TimeSK, Amount, Quantity}.
* Таблицы измерений (нормализованные):
  + "Клиенты": {CustomerSK, CustomerID, Name, CityID} → "Города": {CityID, City, CountryID} → "Страны": {CountryID, Country}.
  + "Продукты": {ProductSK, ProductID, Name, CategoryID} → "Категории": {CategoryID, Category}.
  + "Время": {TimeSK, Date, MonthID} → "Месяцы": {MonthID, Month, Year}.

**Особенности:**

* Нормализованные таблицы измерений, что уменьшает избыточность данных.
* Более сложная структура из-за дополнительных таблиц и связей.
* Используется в случаях, когда важна экономия места или строгая нормализация данных.

#### 3. **Различия между схемой "Звезда" и "Снежинка"**

| **Характеристика** | **Звезда (Star Schema)** | **Снежинка (Snowflake Schema)** |
| --- | --- | --- |
| **Структура** | Денормализованная, простая, плоские таблицы измерений | Нормализованная, иерархическая, много подтаблиц |
| **Количество таблиц** | Меньше таблиц (одна таблица на измерение) | Больше таблиц из-за нормализации измерений |
| **Сложность запросов** | Простые запросы, меньше соединений (JOIN) | Сложные запросы, больше соединений (JOIN) |
| **Производительность** | Высокая, так как меньше соединений и проще индексация | Ниже из-за дополнительных соединений |
| **Объём хранения** | Больше из-за избыточности в денормализованных таблицах | Меньше за счёт нормализации |
| **Гибкость изменений** | Менее гибкая при изменении структуры измерений | Более гибкая для изменений в измерениях |
| **Простота понимания** | Простая для пользователей и разработчиков BI | Более сложная из-за нормализованной структуры |

#### 4. **Преимущества и недостатки схемы "Звезда"**

**Преимущества:**

* **Простота**: Легко понять и использовать для аналитиков и BI-инструментов.
* **Высокая производительность**: Меньше соединений (JOIN) в запросах, что ускоряет выполнение аналитических запросов.
* **Оптимизация для аналитики**: Подходит для типичных запросов в DWH, таких как агрегация, фильтрация и группировка.
* **Простота ETL**: Денормализованные таблицы упрощают процессы извлечения, трансформации и загрузки данных.

**Недостатки:**

* **Избыточность данных**: Денормализация приводит к дублированию данных в таблицах измерений, что увеличивает объём хранения.
* **Ограниченная гибкость**: Изменение структуры измерений (например, добавление новых атрибутов) может потребовать значительных изменений в схеме.
* **Не подходит для сложных иерархий**: Если измерения имеют сложные иерархии, схема "Звезда" становится менее эффективной.

#### 5. **Преимущества и недостатки схемы "Снежинка"**

**Преимущества:**

* **Экономия места**: Нормализация устраняет избыточность, уменьшая объём хранимых данных.
* **Гибкость**: Легче адаптировать к изменениям в структуре измерений (например, добавление новых уровней иерархии).
* **Поддержка сложных иерархий**: Подходит для данных с глубокими иерархическими структурами (например, категории → подкатегории → бренды).

**Недостатки:**

* **Сложность запросов**: Требуется больше соединений (JOIN), что замедляет выполнение запросов, особенно в больших DWH.
* **Сложность для пользователей**: Нормализованная структура сложнее для понимания аналитиками и разработчиками BI-отчётов.
* **Сложность ETL**: Нормализация усложняет процессы загрузки данных, так как требуется поддерживать связи между подтаблицами.
* **Нижняя производительность**: Дополнительные соединения увеличивают время обработки запросов.

#### 6. **Когда использовать "Звезду" или "Снежинку"?**

* **Схема "Звезда"**:
  + Подходит для большинства DWH, где приоритет — производительность и простота запросов.
  + Идеальна для BI-отчётов, аналитики и работы с большими объёмами данных.
  + Используется, когда объём хранения не является критическим фактором.
* **Схема "Снежинка"**:
  + Используется, когда важна экономия пространства или данные имеют сложные иерархии.
  + Подходит для случаев, где требуется высокая нормализация и поддержка сложных изменений в измерениях.
  + Применяется в системах с ограниченными ресурсами хранения или сложной структурой данных.

#### 7. **Пример в контексте**

**Сценарий**: Хранилище данных для розничной сети.

* **Звезда**:
  + Таблица фактов "Продажи": {SaleID, CustomerSK, ProductSK, TimeSK, Amount}.
  + Таблицы измерений: "Клиенты" (все данные о клиенте в одной таблице), "Продукты" (все данные о продукте в одной таблице).
  + Запрос для выручки по городам: один JOIN между таблицей фактов и таблицей "Клиенты".
* **Снежинка**:
  + Таблица фактов та же: {SaleID, CustomerSK, ProductSK, TimeSK, Amount}.
  + Таблица измерений "Клиенты" разбита: "Клиенты" → "Города" → "Страны".
  + Запрос для выручки по городам: несколько JOIN (между "Продажи", "Клиенты", "Города"), что усложняет запрос.

#### 8. **Связь с другими темами**

* **Суррогатные ключи**: Используются в обеих схемах для связи таблиц фактов и измерений. В "Снежинке" их больше из-за дополнительных таблиц.
* **Гранулярность**: Влияет на таблицу фактов. Например, в "Звезде" высокая гранулярность (по транзакциям) упрощает запросы, но увеличивает объём данных.
* **Меры**: Агрегируемые меры (например, выручка) хорошо работают в обеих схемах, но в "Снежинке" сложнее обрабатывать полу-агрегируемые меры из-за нормализации.

#### 9. **Заключение**

Схема "Звезда" — это простой и производительный выбор для большинства хранилищ данных, оптимизированный для аналитики и BI. Схема "Снежинка" подходит для случаев, где важна экономия пространства или сложные иерархии, но она сложнее в реализации и менее производительна. Выбор между ними зависит от целей DWH, объёма данных, требований к производительности и сложности измерений.

## SCD, зачем нужно и какие виды бывают.

**Что такое SCD (Slowly Changing Dimensions)?**

Slowly Changing Dimensions (SCD, медленно изменяющиеся измерения) — это концепция в хранилищах данных (Data Warehouse), которая используется для управления изменениями данных в измерениях (dimensions) с течением времени. Измерения — это справочные таблицы, содержащие описательные атрибуты (например, данные о клиентах, продуктах, географии), которые используются для анализа фактов. Поскольку данные в измерениях могут меняться (например, клиент сменил адрес или фамилию), SCD определяет, как эти изменения обрабатываются, чтобы сохранить историческую целостность и обеспечить точный анализ.

**Зачем нужны SCD?**

SCD необходимы для:

1. **Сохранения исторических данных**: Позволяют отслеживать изменения в данных измерений, чтобы анализировать исторические тренды и корректно связывать факты с соответствющими версиями измерений.
2. **Обеспечения точности аналитики**: Без учета изменений в измерениях отчеты могут быть неточными, так как старые данные могут не соответствовать текущим значениям.
3. **Поддержки бизнес-анализа**: Например, для анализа продаж важно знать, какие атрибуты клиента (регион, категория) были актуальны на момент транзакции.
4. **Гибкости в моделировании данных**: SCD позволяют хранить данные в зависимости от требований бизнеса — от полного сохранения истории до перезаписи данных.

**Виды SCD**

Существует несколько типов SCD, которые определяют, как обрабатываются изменения в измерениях. Основные из них:

1. **SCD Type 0 (Фиксированные данные)**
   * **Описание**: Данные в измерении не изменяются после их первоначальной загрузки. Любые изменения игнорируются.
   * **Применение**: Используется для статичных данных, которые не должны меняться (например, коды стран или фиксированные категории).
   * **Пример**: Таблица с кодами регионов, которые никогда не обновляются.
   * **Особенности**: Простота реализации, но отсутствие гибкости при изменениях.
2. **SCD Type 1 (Перезапись)**
   * **Описание**: При изменении данных старая информация перезаписывается новой без сохранения истории.
   * **Применение**: Используется, когда история изменений не важна, а актуальность данных приоритетна (например, текущий адрес клиента).
   * **Пример**: Клиент сменил адрес с "ул. Ленина" на "ул. Мира". В таблице старый адрес заменяется новым.
   * **Особенности**: Простота, минимальный объем данных, но потеря истории.
3. **SCD Type 2 (Добавление новой записи)**
   * **Описание**: При изменении данных создается новая запись с обновленными значениями, а старая помечается как устаревшая (обычно с помощью дат действия или флага).
   * **Применение**: Используется, когда важно сохранить полную историю изменений (например, для отслеживания смены статуса клиента).
   * **Пример**: Клиент сменил статус с "Стандарт" на "Премиум". Создается новая запись с новым статусом и датой начала действия, а старая сохраняется с датой окончания.
   * **Особенности**: Полное сохранение истории, но увеличение объема данных. Требуются поля для дат действия (Valid From, Valid To) или флага активности.
4. **SCD Type 3 (Частичное сохранение истории)**
   * **Описание**: Сохраняется ограниченная история изменений путем добавления столбцов для предыдущих значений (например, "старое значение" и "новое значение").
   * **Применение**: Используется, когда нужно хранить только последнее изменение или ограниченное количество предыдущих значений.
   * **Пример**: Клиент сменил регион. В таблице добавляется столбец "Предыдущий регион" для хранения старого значения.
   * **Особенности**: Компромисс между Type 1 и Type 2, но ограничен количеством хранимых изменений.
5. **SCD Type 4 (Таблица истории)**
   * **Описание**: Текущие данные хранятся в основной таблице измерений, а исторические — в отдельной таблице истории.
   * **Применение**: Используется для оптимизации производительности, когда текущие данные часто запрашиваются, а история — редко.
   * **Пример**: Основная таблица содержит актуальный адрес клиента, а таблица истории — все предыдущие адреса с датами.
   * **Особенности**: Уменьшает объем основной таблицы, но требует управления двумя таблицами.
6. **SCD Type 6 (Гибридный подход)**
   * **Описание**: Комбинация Type 1, Type 2 и Type 3. Для каждой записи хранится текущая и историческая информация с использованием флагов, дат и дополнительных столбцов.
   * **Применение**: Используется в сложных системах, где требуется максимальная гибкость.
   * **Пример**: Таблица хранит текущий статус клиента (Type 1), историю изменений статуса (Type 2) и предыдущее значение статуса в отдельном столбце (Type 3).
   * **Особенности**: Сложность реализации, но высокая гибкость.

**Заключение**

SCD — ключевой элемент в проектировании хранилищ данных, который позволяет эффективно управлять изменениями в измерениях. Выбор типа SCD зависит от бизнес-требований: Type 1 подходит для простоты и актуальности, Type 2 — для сохранения полной истории, Type 3 и 4 — для компромиссных решений, а Type 6 — для сложных сценариев. Правильное использование SCD обеспечивает точность аналитики и поддержку исторических данных, что критично для принятия бизнес-решений.

## Инкрементальный подход загрузки данных в DWH.

### 1. Инкрементальный подход в DWH

**Что это?**  
Инкрементальный подход в хранилищах данных — это метод загрузки данных, при котором в DWH загружаются только новые или измененные данные, а не весь объем данных из источников. Это противоположность полному (full load) подходу, где каждый раз загружается весь набор данных.

**Зачем нужен?**

* Ускоряет процесс загрузки, так как обрабатывается только дельта (новые или измененные записи).
* Снижает нагрузку на источники данных и хранилище.
* Экономит ресурсы (время, память, вычислительные мощности).

**Как это работает?**

* Источник данных отслеживает изменения (например, новые записи, обновления, удаления).
* Эти изменения извлекаются и загружаются в DWH с помощью ETL/ELT-процессов.
* Для определения изменений используются методы, такие как CDC, временные метки или хэш-значения.

### 2. Методы инкрементальной загрузки

Существует несколько методов, чтобы определить, какие данные изменились и должны быть загружены. Основные из них:

#### a) Временные метки (Timestamps)

**Что это?**  
Временные метки — это столбцы в таблице источника, которые показывают, когда запись была создана или обновлена (например, created\_at, updated\_at).

**Как работает?**

* В DWH хранится информация о последней обработанной временной метке (например, максимальная updated\_at).
* При следующей загрузке выбираются только записи, где updated\_at больше этой метки.
* Загружаются новые или измененные записи.

**Плюсы:**

* Простота реализации.
* Хорошо работает, если в источнике есть надежные метки времени.

**Минусы:**

* Не отслеживает удаленные записи (нужен дополнительный механизм).
* Требует, чтобы источник данных поддерживал временные метки.
* Проблемы с часовыми поясами или некорректными метками могут сломать процесс.

**Пример:**  
В таблице заказов есть столбец updated\_at. Последняя загрузка была 2025-05-29 23:59:59. В следующий раз ETL-процесс выберет все записи, где updated\_at > 2025-05-29 23:59:59, и загрузит их в STG или ODS.

#### b) CDC (Change Data Capture — захват изменений данных)

**Что это?**  
CDC — это технология или подход, который отслеживает изменения в данных (вставки, обновления, удаления) на уровне источника, обычно с помощью логов базы данных или триггеров.

**Как работает?**

* Источник (например, база данных) фиксирует изменения в специальном логе (например, журнал транзакций в SQL Server или Oracle).
* ETL-инструмент (например, Apache Kafka, Debezium) читает лог и передает изменения в DWH.
* CDC может работать в реальном времени или по расписанию.

**Плюсы:**

* Точно отслеживает все типы изменений (вставки, обновления, удаления).
* Подходит для систем с высокой нагрузкой и реального времени.
* Минимизирует вмешательство в источник данных.

**Минусы:**

* Требует поддержки CDC на уровне источника (не все базы данных это поддерживают).
* Сложнее в настройке, чем временные метки.
* Может быть ресурсоемким для больших систем.

**Пример:**  
В базе данных PostgreSQL включен CDC через расширение wal2json. ETL-инструмент читает лог изменений и загружает только новые заказы или обновленные статусы заказов в DWH.

#### c) Хэш-значения (Hash Values)

**Что это?**  
Хэш-значение — это уникальная строка, которая генерируется на основе содержимого записи с помощью хэш-функции (например, MD5, SHA-1, SHA-256). Она используется для сравнения записей и выявления изменений.

**Как работает?**

* Для каждой записи в источнике вычисляется хэш на основе ключевых столбцов (например, CONCAT(column1, column2, ...) → SHA256()).
* Хэш сохраняется в DWH вместе с данными.
* При следующей загрузке хэш новой версии записи сравнивается с сохраненным. Если хэш изменился, запись считается обновленной и загружается.

**Плюсы:**

* Не зависит от временных меток или поддержки CDC в источнике.
* Позволяет точно определять изменения в данных.
* Может использоваться для сложных данных, где нет явных меток времени.

**Минусы:**

* Вычисление хэшей требует дополнительных ресурсов.
* Не отслеживает удаленные записи (нужен дополнительный механизм).
* Требует хранения хэшей в DWH, что увеличивает объем данных.

**Пример:**  
В таблице клиентов есть поля name, email, phone. Для каждой записи вычисляется хэш: SHA256(name || email || phone). Если хэш новой версии записи отличается от старого, запись обновляется в DWH.

#### d) Полное сравнение (Full Compare)

**Что это?**  
Полное сравнение предполагает загрузку всех данных из источника в промежуточную таблицу (например, в STG) и их последующее сравнение с данными в DWH.

**Как работает?**

* Все данные из источника загружаются в STG.
* Сравниваются записи (по ключу) между STG и DWH.
* Определяются новые, обновленные или удаленные записи и применяются изменения.

**Плюсы:**

* Надежный способ, так как проверяет все данные.
* Отслеживает все типы изменений (включая удаления).

**Минусы:**

* Ресурсоемкий, так как требует загрузки и сравнения больших объемов данных.
* Медленнее, чем другие методы.

**Пример:**  
Таблица заказов из источника полностью загружается в STG. Затем SQL-запрос сравнивает order\_id и поля данных между STG и DDS, обновляя только измененные записи.

#### e) Логические флаги или идентификаторы изменений

**Что это?**  
Источник данных содержит флаг или идентификатор, который указывает, была ли запись изменена (например, столбец is\_modified или change\_id).

**Как работает?**

* ETL-процесс выбирает записи, где флаг указывает на изменение (is\_modified = 1).
* После загрузки флаг сбрасывается или обновляется идентификатор.

**Плюсы:**

* Простота реализации, если источник поддерживает флаги.
* Минимальная нагрузка на ETL.

**Минусы:**

* Требует модификации источника данных.
* Не всегда отслеживает удаления.

**Пример:**  
В таблице источника есть столбец is\_modified. ETL загружает только записи с is\_modified = 1 и сбрасывает флаг после обработки.

### 3. Что такое хэш-значение?

**Определение:**  
Хэш-значение — это строка фиксированной длины, которая генерируется из входных данных с помощью хэш-функции. Хэш-функция преобразует данные (например, строку или набор столбцов) в уникальный код, который чувствителен к любым изменениям во входных данных.

**Как используется в DWH?**

* Для инкрементальной загрузки: сравнение хэшей позволяет быстро определить, изменилась ли запись.
* Для дедупликации: одинаковые хэши указывают на одинаковые данные.
* Для проверки целостности данных: хэш подтверждает, что данные не были случайно изменены.

**Пример:**  
Запись {name: "Иван", email: "ivan@example.com"} преобразуется в хэш SHA256("Иванivan@example.com") = "a1b2c3...". Если email изменится на ivan2@example.com, хэш станет другим, что сигнализирует об изменении.

**Популярные хэш-функции:**

* MD5 (быстрая, но менее безопасная).
* SHA-1, SHA-256 (более надежные).
* CRC32 (простая, но только для базовых задач).

### 4. CDC и временные метки в контексте DWH

**CDC (Change Data Capture):**

* CDC часто используется для инкрементальной загрузки в реальном времени.
* Пример: Debezium читает журнал транзакций MySQL и отправляет изменения в Kafka, откуда они загружаются в DWH.
* CDC может комбинироваться с хэшами или временными метками для большей надежности. Например, CDC определяет, какие записи изменились, а хэш подтверждает, какие поля в записи обновились.

**Временные метки:**

* Часто используются в паре с CDC, если источник не поддерживает сложные логи изменений.
* Пример: CDC фиксирует, что запись была обновлена, а временная метка (updated\_at) помогает отфильтровать только свежие изменения.
* Временные метки также используются для построения исторических данных в DWH (например, в моделях Slowly Changing Dimensions, SCD).

### 5. Почему именно эти методы?

* **Временные метки** просты и подходят для баз с явными метками времени, но не всегда надежны для сложных систем.
* **CDC** идеален для реального времени и сложных баз, но требует инфраструктуры.
* **Хэш-значения** универсальны и точны для определения изменений, но увеличивают сложность ETL.
* **Полное сравнение** надежно, но медленно и ресурсоемко.
* **Логические флаги** удобны, если источник поддерживает их, но требуют вмешательства в источник.

Выбор метода зависит от:

* Типа источника данных (поддерживает ли CDC, есть ли метки времени).
* Объема данных и частоты обновлений.
* Требований к скорости и точности.

### 6. Пример интеграции

Допустим, у нас есть интернет-магазин, и мы загружаем данные о заказах в DWH:

1. **Row/STG:** Данные о заказах выгружаются из базы в виде JSON и сохраняются в Raw. В STG они загружаются с проверкой формата.
2. **CDC:** Debezium отслеживает изменения в базе (новые заказы, изменения статуса).
3. **Временные метки:** ETL выбирает заказы, где updated\_at новее последней загрузки.
4. **Хэш-значения:** Для каждой записи вычисляется хэш на основе order\_id, customer\_id, amount. Если хэш изменился, запись обновляется в DDS.
5. **Data Mart:** В витрине данных создается агрегированная таблица с метриками по продажам для аналитиков.

## Витрина данных. Зависимые и независимые витрины.

**Витрина данных (Data Mart): Зависимые и независимые витрины**

**Что такое витрина данных?**

Витрина данных (Data Mart) — это специализированное подмножество хранилища данных (Data Warehouse, DWH), предназначенное для удовлетворения аналитических потребностей конкретного отдела, группы пользователей или бизнес-процесса. Витрина содержит агрегированные, структурированные и оптимизированные данные, которые ориентированы на определенные аналитические задачи, такие как отчетность, дашборды или углубленный анализ.

В отличие от DWH, которое хранит данные в централизованном виде для всей организации, витрина данных фокусируется на узком наборе данных, упрощая доступ и повышая производительность запросов для конечных пользователей.

**Зачем нужны витрины данных?**

1. **Упрощение доступа к данным**: Витрины предоставляют пользователям (аналитикам, менеджерам) данные в удобном, заранее подготовленном формате, снижая сложность работы с DWH.
2. **Оптимизация производительности**: Данные в витрине агрегированы и оптимизированы для конкретных запросов, что ускоряет выполнение отчетов и аналитики.
3. **Поддержка бизнес-процессов**: Витрины создаются под нужды конкретных отделов (финансы, маркетинг, продажи), что делает их более релевантными для пользователей.
4. **Снижение нагрузки на DWH**: Запросы пользователей перенаправляются на витрины, что уменьшает нагрузку на основное хранилище.
5. **Гибкость и масштабируемость**: Позволяют адаптировать данные под разные аналитические задачи без изменения структуры DWH.

**Типы витрин данных: Зависимые и независимые**

Витрины данных классифицируются на зависимые и независимые в зависимости от их связи с центральным хранилищем данных.

1. **Зависимые витрины данных**
   * **Описание**: Зависимая витрина создается как подмножество данных, извлеченных из центрального DWH. Она полностью зависит от структуры и данных хранилища, а данные в ней обычно синхронизированы с DWH.
   * **Характеристики**:
     + Данные извлекаются из DWH с помощью ETL-процессов.
     + Обеспечивается единая точка истины (single source of truth), так как данные согласованы с DWH.
     + Структура витрины определяется моделью данных DWH.
   * **Применение**: Используется в крупных организациях с централизованным DWH, где требуется единообразие данных (например, финансовая отчетность).
   * **Пример**: Витрина для отдела продаж, содержащая агрегированные данные о заказах, клиентах и выручке, извлеченные из DWH.
   * **Преимущества**:
     + Согласованность данных с DWH.
     + Централизованное управление и контроль качества данных.
   * **Недостатки**:
     + Зависимость от DWH может ограничивать гибкость.
     + Требуется доступ к DWH, что может замедлить разработку витрины.
2. **Независимые витрины данных**
   * **Описание**: Независимая витрина создается без привязки к центральному DWH, данные загружаются напрямую из источников (например, оперативных систем, баз данных, внешних API).
   * **Характеристики**:
     + Данные собираются и обрабатываются независимо от DWH.
     + Структура и модель данных витрины определяются конкретными потребностями пользователей.
     + Может существовать отдельно от DWH или вообще без него.
   * **Применение**: Используется в небольших компаниях или для специфических задач, где нет необходимости в централизованном DWH, либо для быстрого прототипирования.
   * **Пример**: Витрина для маркетингового отдела, собирающая данные о кампаниях из CRM-системы и Google Analytics без интеграции с DWH.
   * **Преимущества**:
     + Высокая гибкость и скорость разработки.
     + Не требует наличия DWH.
   * **Недостатки**:
     + Риск несогласованности данных между разными витринами.
     + Сложность в обеспечении единой точки истины.
     + Дублирование данных и ETL-процессов.

**Сравнение зависимых и независимых витрин**

| **Характеристика** | **Зависимая витрина** | **Независимая витрина** |
| --- | --- | --- |
| **Источник данных** | Центральное DWH | Оперативные системы, внешние источники |
| **Согласованность** | Высокая (единая точка истины) | Низкая (возможны расхождения) |
| **Скорость создания** | Медленнее (зависит от DWH) | Быстрее (автономная разработка) |
| **Гибкость** | Ограничена структурой DWH | Высокая, адаптируется под задачу |
| **Обслуживание** | Централизованное | Децентрализованное, сложнее |

**Примеры использования**

1. **Зависимая витрина**:
   * **Сценарий**: Финансовый отдел крупной компании хочет анализировать квартальную выручку по регионам.
   * **Реализация**: Создается витрина на основе DWH, содержащая агрегированные данные о продажах, клиентах и географии. ETL-процесс извлекает данные из таблиц фактов и измерений DWH, применяя фильтры и агрегации. Витрина используется для построения дашбордов в BI-инструменте (например, Power BI).
   * **Инструменты**: SQL, Informatica, Talend.
2. **Независимая витрина**:
   * **Сценарий**: Маркетинговый отдел небольшой компании анализирует эффективность рекламных кампаний.
   * **Реализация**: Данные собираются напрямую из Google Ads и CRM (например, HubSpot) в отдельную витрину, где они агрегируются для анализа ROI кампаний. DWH отсутствует, так как компания небольшая.
   * **Инструменты**: Apache Airflow, Python, Google BigQuery.

**Заключение**

Витрины данных — это эффективный инструмент для предоставления пользователям удобного доступа к данным для аналитики и отчетности. Зависимые витрины подходят для организаций с централизованным DWH, обеспечивая согласованность и контроль, но требуют больше времени на разработку. Независимые витрины предлагают гибкость и скорость, но могут привести к проблемам с согласованностью данных. Выбор типа витрины зависит от масштаба компании, архитектуры данных и бизнес-требований.

## Какие области хранилища существуют и для чего каждая из них нужна

### 1. STG (Staging Area — промежуточный слой)

**Что это?**  
STG — это временное хранилище, куда данные поступают из источников (например, базы данных, API, файлы) в их "сыром" виде. Это первый слой, куда данные загружаются перед дальнейшей обработкой.

**Для чего нужен?**

* Служит "прихожей" для данных, чтобы не ломать основное хранилище, если что-то пойдет не так.
* Хранит данные в исходном формате, чтобы можно было к ним вернуться для проверки или повторной обработки.
* Используется для минимальной очистки данных (например, удаление дубликатов, приведение типов).

**Что происходит?**

* Данные загружаются из источников (например, CSV, JSON, SQL-таблиц).
* Выполняется базовая валидация (проверка на целостность, формат).
* Данные хранятся временно, часто в том же формате, что и в источнике.
* Никакой сложной логики или трансформации тут обычно нет.

**Почему так?**  
STG нужен, чтобы изолировать "грязные" данные от основного хранилища. Если данные содержат ошибки, их можно исправить на этом этапе, не затрагивая основные слои. Это как черновик перед чистовиком.

**Пример:**  
Из CRM-системы выгружаются данные о продажах в виде CSV. В STG они загружаются как есть, но проверяется, что все строки имеют нужное количество столбцов и нет пустых значений в ключевых полях.

### 2. ODS (Operational Data Store — операционное хранилище данных)

**Что это?**  
ODS — это слой, где данные из STG собираются, интегрируются и хранятся в более структурированном виде, но всё ещё близком к исходным системам. Это "оперативная" база данных для текущих данных.

**Для чего нужен?**

* Хранит актуальные данные для оперативной аналитики (например, отчеты за день или неделю).
* Обеспечивает интеграцию данных из разных источников в единую структуру.
* Используется для задач, требующих быстрого доступа к текущим данным (например, мониторинг транзакций).

**Что происходит?**

* Данные из STG очищаются и приводятся к единому формату (например, унифицируются даты, коды).
* Данные обновляются регулярно (например, ежедневно), отражая текущее состояние источников.
* ODS может поддерживать транзакции и быстрые запросы, как в обычной базе данных.

**Почему так?**  
ODS нужен для оперативной работы, когда данные должны быть доступны быстро и в удобном виде, но ещё не требуют сложной аналитической обработки. Это как "рабочий стол", где всё организовано, но ещё не разложено по полочкам для долгосрочного хранения.

**Пример:**  
В ODS хранятся данные о заказах интернет-магазина за последние 30 дней: ID заказа, клиент, сумма, дата. Эти данные обновляются каждый день и используются для построения оперативных отчетов.

### 3. DDS (Data Delivery Store — слой доставки данных, или Core Data Warehouse)

**Что это?**  
DDS — это центральный слой хранилища данных, где данные хранятся в нормализованном виде, оптимизированном для аналитики. Это "ядро" хранилища, где данные структурированы для долгосрочного хранения и анализа.

**Для чего нужен?**

* Хранит исторические данные в структурированном виде для глубокого анализа.
* Поддерживает сложные аналитические запросы (например, тренды за годы).
* Обеспечивает целостность и согласованность данных.

**Что происходит?**

* Данные из ODS (или напрямую из STG) трансформируются в нормализованные таблицы (например, по модели "звезда" или "снежинка").
* Выполняется глубокая очистка, дедупликация, обогащение (например, добавление категорий).
* Данные агрегируются и хранятся в формате, удобном для аналитики (например, факты и измерения).

**Почему так?**  
DDS — это "библиотека" хранилища, где данные организованы для долгосрочного хранения и сложных запросов. Нормализация уменьшает избыточность и облегчает анализ. Этот слой нужен, чтобы аналитики могли работать с историческими данными, не перегружая оперативные системы.

**Пример:**  
В DDS хранится таблица фактов с продажами (дата, товар, сумма) и таблицы измерений (информация о товарах, клиентах). Это позволяет строить отчеты вроде "продажи по категориям за 5 лет".

### 4. Row (Raw Data Layer — слой необработанных данных)

**Что это?**  
Row (или Raw Data Layer) — это слой, где хранятся необработанные данные в их исходном виде, часто в виде "сырых" логов или архивов. Иногда этот слой путают со STG, но он обычно предназначен для долгосрочного хранения.

**Для чего нужен?**

* Хранит полные копии данных из источников для архивации.
* Используется для восстановления данных или повторной обработки, если что-то пошло не так.
* Может быть источником для новых аналитических задач, требующих исходных данных.

**Что происходит?**

* Данные загружаются из источников и сохраняются без изменений.
* Никакой очистки или трансформации не проводится.
* Данные могут храниться в формате файлов (например, Parquet, Avro) или в NoSQL-базах.

**Почему так?**  
Row нужен как "архивный склад", чтобы сохранить исходные данные на случай, если потребуется вернуться к ним (например, если в STG или ODS что-то сломалось). Это страховка от потери данных.

**Пример:**  
Логи транзакций из платежной системы хранятся в Row в виде JSON-файлов. Если аналитику нужно изучить редкий тип транзакций, он обращается к Row, а не к обработанным данным.

### 5. Data Mart (витрины данных)

**Что это?**  
Data Mart — это специализированный слой, где данные из DDS агрегируются и оптимизируются для конкретных аналитических задач или отделов (например, маркетинга, финансов).

**Для чего нужен?**

* Обеспечивает быстрый доступ к данным для конкретных бизнес-потребностей.
* Упрощает аналитику за счет предобработанных и агрегированных данных.
* Часто оптимизирован для работы с BI-инструментами (например, Tableau, Power BI).

**Что происходит?**

* Данные из DDS агрегируются и преобразуются в формат, удобный для конкретного отдела (например, сводные таблицы).
* Создаются представления (views) или материализованные таблицы с нужными метриками.
* Данные денормализуются для ускорения запросов.

**Почему так?**  
Data Mart — это как "готовые блюда" для бизнеса. Вместо того чтобы каждый раз копаться в DDS, пользователи получают готовые наборы данных, оптимизированные под их задачи. Это экономит время и ресурсы.

**Пример:**  
Data Mart для маркетинга содержит агрегированные данные о кампаниях: бюджет, клики, конверсии. Маркетологи используют его для построения дашбордов в Power BI.

### 6. Метаданные (Metadata)

**Что это?**  
Метаданные — это "данные о данных", которые описывают структуру, происхождение и трансформации данных в хранилище. Это не слой в прямом смысле, а скорее система управления информацией о хранилище.

**Для чего нужны?**

* Помогают понять, откуда пришли данные, как они обрабатывались и где хранятся.
* Обеспечивают управление хранилищем (например, контроль версий, аудит).
* Упрощают поиск и использование данных (каталог данных).

**Что происходит?**

* Хранятся описания таблиц, столбцов, их типов, источников данных.
* Фиксируются правила трансформации (например, как данные из STG переходят в DDS).
* Используются для автоматизации процессов (например, в инструментах ETL).

**Почему так?**  
Метаданные — это как "инструкция" к хранилищу. Без них аналитики и разработчики теряются в сложной структуре данных. Они обеспечивают прозрачность и управляемость.

**Пример:**  
Метаданные содержат информацию, что таблица "Sales" в DDS создана из таблиц STG "Orders" и "Customers", с указанием, какие столбцы были объединены и какие правила применялись.

### Почему архитектура именно такая?

Эта многослойная структура (STG → ODS → DDS → Data Mart) существует, чтобы разделить задачи обработки данных:

* **STG** защищает хранилище от "грязных" данных.
* **ODS** обеспечивает оперативный доступ к текущим данным.
* **DDS** хранит исторические данные для глубокого анализа.
* **Row** сохраняет исходные данные для надежности.
* **Data Mart** упрощает доступ для конечных пользователей.
* **Метаданные** делают систему прозрачной и управляемой.

Каждый слой решает свою задачу, минимизируя хаос и обеспечивая эффективность. Это как в ресторане: сырые продукты (Row, STG), подготовка ингредиентов (ODS), готовка блюд (DDS) и подача на стол (Data Mart).